UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

RECOLHA E ANÁLISE DE DADOS EM AMBIENTES HOSTIS

Cláudio Miguel Hotelão de Oliveira Saramagaio

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Engenharia de Software

2014

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

RECOLHA E ANÁLISE DE DADOS EM AMBIENTES HOSTIS

Cláudio Miguel Hotelão de Oliveira Saramagaio

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Engenharia de Software

Dissertação orientada pelo Prof. Doutor Francisco Cipriano da Cunha Martins

2014

Agradecimentos

Agradeço em primeiro lugar ao meu orientador, Professor Doutor Francisco Martins, por toda ajuda durante este ano, tendo estado sempre disponível para ajudar a entender e resolver todos os problemas que surgiram durante o desenvolvimento do projeto. Ao LaSIGE por me ter acolhido durante este último ano, e me ter proporcionado a possibilidade de investigar e criar algo novo. Ao Carlos Machado do Departamento de Ciência de Computação da Faculdade de Ciências da Universidade de Porto, pela ajuda prestada na parte de eletrónica e sensores, áreas estas que eu não dominava no início do projeto e hoje, graças à sua ajuda, consigo entendê-las.

Quero também agradecer à minha família, porque sem ela nada disto seria possível, pois foi quem mais contribuiu para eu ser aquilo que sou hoje e ter chegado onde cheguei. Além da família, tenho também a agradecer aos meus amigos. Àqueles que já fazem parte da minha vida há alguns anos, por também me terem ajudado a formar a como pessoa, bem como aos amigos da faculdade, sem os quais nada disto seria possível, tendo me ajudado a ultrapassar todas barreiras ao longo da vida académica. Não me esquecendo da minha namorada, tenho também a agradecer-lhe por todo o apoio prestado e compreensão por nem sempre estar disponível.

Por último, tenho a agradecer aos Bombeiros Voluntários de Vendas Novas. Para muitos uma associação ou instituição, para outros uma casa, da qual faço parte já há alguns anos. Esta ajudou-me a crescer e a enfrentar alguns problemas da vida. Nestes últimos meses do projeto, possibilitou-me também efetuar algum trabalho relacionado com o mesmo.

A todos, um grande OBRIGADO!

Para família e amigos verdadeiros.

Resumo

O projeto insere-se no tema "A Internet das Coisas". Tem como objetivo criar um *middleware* que possa ser executado por dispositivos com baixas capacidades de processamento e que garanta um baixo consumo de recursos. Este *middleware* terá de ser capaz de agregar informações de redes de sensores e disponibilizar serviços a aplicações cliente. O sistema final será aplicado para monitorizar o trabalho dos bombeiros na área dos incêndios florestais. Terá como base a monitorização de gases e as temperaturas a que os bombeiros estão expostos num cenário de um incêndio florestal. Procede também à localização através de GPS, a fim de efetuar cálculos acerca do desgaste dos bombeiros, e os localizar em caso de acidente. Para tal construíu-se uma rede de sensores, que conseguirá captar os dados do ambiente que se pretendem. Para os agregar e disponibilizar desenvolveu-se um *middleware* para operar nas estações base e armazenar os dados recolhidos pelas redes de sensores. Por fim desenvolveu-se uma aplicação cliente para monitorizar os bombeiros envolvidos, que usufrui de *web services* disponibilizados pelo *middleware*. De forma a criar a rede de sensores, foram construídos dispositivos que incluem sensores de gases e temperatura bem como uma módulo para localização de GPS e uma placa de rede ZigBee, os quais são alimentados por uma pilha. O *middleware* desenvolvido pode ser expandido em tempo de execução adcionando novos *gateways* ou serviços. Esse *gateways* tem como objetivo agregra de redes de sensores com diferentes tipos de tecnologias. Em relação aos serviços estes podem coomunicar com outros, serviços, *gateways* ou outros componentes do sistema, a fim de implementar uma nova funcionalidade ou efetuar cálculos a partir dos dados recolhidos. A aplicaçao desenvolvida, tem como pano de fundo as tecnologias *web*, consiste numa aplicação híbrida. Esta pode ser executada em computadores, recorrendo a um *web browser*, ou exportada para plataformas móveis.

Palavras-chave: *middleware*, sensores, gases, redes de sensores, incêndios florestais, bombeiros

Abstract

This project fits into the ”The Internet of Things” theme. It aims at creating a middleware capable of aggregating sensor networks and of providing services to client applications, ensuring low resources consumption and the possibility to run on devices with low processing capacity. The final system is applied to firefighters missions, particularly in fighting forest fires. Its purpose is to monitor toxic gases and temperatures. That firefighters are exposed during the fight of forest fires, as well as to locate each firefighter via GPS in order to calculate their physical tiredness and locate them in case of accidents. For this we create a sensor network, that collects environmental data. In order to ensure that data are available to client applications we developed a middleware to be run on base stations, which stores data collected by sensor networks and make them available to clients on web services. We also developed a client application in order to monitor firefighters involved in forest fire, which makes use of the web services provided by the middleware.

Keywords: middleware, sensor networks, sensors, wildfire, gas, firefighter

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Um dos grandes desafios tecnológicos dos nossos dias é a descoberta e o desenvolvimento de dispositivos de baixo custo suficientemente poderosos para se ligarem à Internet, e que ao mesmo tempo garantam baixos consumos de energia. Estes dispositivos são a chave para A Internet das Coisas (IoT, do inglês *The Internet of Things*) [35].

A Internet das Coisas representa uma revolução tecnológica e dá-nos uma antevisão do futuro das sociedades da informação. Este novo paradigma permite formas de interação inovadoras com o ambiente que nos rodeia, estando dependente da invoção e desenvolvimento de mecanismos de comunicação sem fios e, ao mesmo tempo, da nanotecnologia. A relação entre estas áreas e A Internet das Coisas advém do seu próprio conceito e objetivo - a interação com o ambiente - medindo parâmetros e tomando decisões baseadas em valores de referência pré-estabelecidos e algoritmos de tomada de decisão. Os valores recolhidos a partir destas medições (por exemplo através de redes de sensores) podem ser armazenados em bases de dados na Internet.

Graças à capacidade de comunicação em locais onde não é possivel estabelecer ligações com fios, ao tamanho reduzido dos dispositivos, e às vantagens que podemos obter numa determinada tarefa ou trabalho com o auxílio da computação, as redes de sensores estão hoje em dia presentes nas mais variadas áreas (*e.g.*, militar, turismo, controlo de armazém, segurança, emergência médica, educação, deteção de catástrofes, entre outras).

A diversidade de áreas às quais as redes de sensores são aplicadas deve-se principalmente à sua versatilidade, uma vez que o mesmo nó da rede tanto pode estar a medir o estado do tempo numa determinada região como a detetar incêndios, dependendo dos sensores que contenha. Da mesma forma, uma estação base (dispositivo que liga redes de sensores, por exemplo, a aplicações cliente ou atuadores) também pode interagir com várias redes simultaneamente, caso esta consiga abstrair o tipo de equipamentos com os quais está a comunicar e o tipo de informações enviadas. Tipicamente esta abstração é conseguida com o uso de *middleware* próprio para este tipo de redes e equipamentos.

O objetivo deste trabalho é desenvolver um sistema para segurança e proteção de bombeiros durante o combate a incêndios florestais utilizando redes de sensores.

O projeto FUMEXP [33], indica que tem sido dada uma grande atenção à exposição de bombeiros a gases durante o combate a incêndios urbanos e industriais, sendo por vezes esquecidas as condições e os perigos inerentes ao combate de incêndios florestais. O estudo revela que existem também bastantes perigos para a saúde humana, em termos de doenças respiratórias, que podem ter origem na exposição a certos tipos de gases e poeiras a que os bombeiros estão expostos durante o combate a incêndios florestais. Depois de três anos de estudo monitorizando gases e poeiras em fogos experimentais, assim como em incêndios florestais, concluiu-se que na realidade da floresta portuguesa os bombeiros estão expostos a altas concentrações de compostos orgânicos voláteis (VOC's, do inglês *Volatile Organic Compounds*), gases tóxicos (*e.g.*, CO, NO_2), e poeiras (*e.g.*, PM10 e PM2.5, que são partículas microscópicas normalmente libertadas pela combustão de materiais orgânicos).

Alguns gases tóxicos (*e.g.* CO) têm efeitos nocivos para a saúde humana quer a curto, quer a longo prazo, podendo muitas vezes ter resultados catastróficos. Por exemplo, o monóxido de carbono (CO) é um gás incolor e inodoro bastante perigoso para a saúde humana, e um humano não se apercebe que está a inalá-lo, podendo facilmente provocar intoxicações. Uma exposição curta a este gás pode provocar sensação de confusão e desorientação, que muitas vezes se agrava com o calor e a baixa visibilidade, comum em cenários de incêndios florestais. Exposições mais prolongadas ao CO podem resultar em toxicidade grave do sistema nervoso central e do coração, podendo mesmo levar à morte. [34, 13].

1.2 Objetivos

O objetivo deste projeto é criar um sistema capaz de medir a temperatura e gases para segurança e proteção dos bombeiros durante o combate a incêndios florestais. Os objetivos do sistema podem ser agrupados em três partes:

1. Construção de **rede de sensores** para capturar dados do ambiente relevantes em cenários de incêndios florestais;

2. Desenvolvimento de um ***middleware*** com a capacidade de ser executado em dispositivos com recursos limitados, capaz de comunicar com redes de sensores ou outros dispositivos móveis e disponibilizar serviços;

3. Construção de uma **aplicação** que mostra as concentrações de gases e a temperatura a que cada bombeiro está exposto, bem como a localização de cada um deles. Esta aplicaçcao tira partido dos serviços oferecidos pelo *middleware* desenvolvido.

De seguida serão detalhados os objetivos para cada uma das partes constituintes do sistema.

1.2.1 Rede de sensores

A rede de sensores tem como principal objetivo recolher valores das concentrações de gases e temperatura que devem ser monitorizados de modo a prevenir acidentes com bombeiros durante o combate a incêndios florestais. A escolha foi feita tendo em conta estudos que revelaram quais os maiores perigos em termos gases tóxicos e poeiras [33]. Optou-se por medir:

- Monóxido de carbono (CO);
- Dioxodo de Azoto (NO_2);
- Poeiras (PM 2.5).

Tendo em conta que existem outros fatores que também afetam o estado físico e psicológico do ser humano decidiu-se monitorizar o conforto térmico, um dos fatores essencias ao bem estar do ser humano [20, 37, 38]. A exposição a valores elevados de radiação de calor pode afetar o corpo humano, fazendo subir a temperatura corporal que podem provocar sérias lesões em casos extremos. Assim, decidiu-se controlar também a temperatura ambiente a que cada indivíduo está exposto.

Como o sistema será aplicado a pessoas cujo esforço físico é demasiado elevado e distribuído heterogeneamente pelas equipas, esta informação é fundamental a quem está a coordenar os trabalhos, pelo que se decidiu incorporar um sensor de GPS. Assim, é possível calcular as distâncias percorridas por cada elemento de forma a ter uma noção do desgaste individual e das equipas. Adicionalmente, o uso de GPS nos dispositivos que acompanham os bombeiros pode ajudar a que estes sejam encontrados rapidamente em caso de acidente.

Abordagem a seguir:

1 Distribuir um dispositivo por cada bombeiro que contém:

  1.1 Um sensor de CO;

  1.3 Um sensor de NO_2;

  1.4 Um sensor de temperatura;

  1.5 Um sensor de localização (GPS);

2 Distribuir por cada viatura um dispositvo que funciona como estação base e que dispõe de:

  2.1 Um sensor de poeiras (PM2.5);

2.2 Um sensor de localização (GPS);

3 Os dispositivos dos bombeiros devem comunicar com as estações base de modo a enviar os dados recolhidos.

1.2.2 *Middleware*

Segundo Karl Aberer *et al.* [15], A Internet das Coisas pretende dotar os objetos da vida quotidiana com capacidade de processamento de informação e comunicação entre eles. Desta forma os computadores poderão medir e apresentar aspetos do mundo físico, bem como reagir em função das medições efetuadas. Contudo, isto cria enormes problemas em termos de heterogeniedade, devido ao vasto número de fontes de dados que necessitam estar conectadas e relacionadas entre si. Uma forma de ultrapssar este problema é utilizar uma camada intermédia, um *middleware* para solucionar alguns dos problemas de heterogeniedade das tecnologias envolvidas.

Por outro lado, para que o sistema efetue as operações pretendidas, é necessário armazenar dados e permitir a comunicação aos dispositivos que recolhem dados ou que pretendem aceder a estes. Assim, tendo em conta o trabalho passado [14, 18, 36, 40] e as necessidades do nosso sistema, pretende-se desenvolver um *middleware* que consiga:

1 Comunicar com um vasto leque de dispositivos

Desafio: Deverá permitir o acesso aos dados independentemente da rede de sensores. Ou seja, independentemente do tipo de dispositivos e de comunicação utilizada por estes (*e.g., ZigBee, Bluetooth*), deverá permitir que estes enviem ou obtenham dados.

Abordagem a seguir: Criar um módulo capaz de abstrair o tipo de rede de onde os dados são provenientes, algo semelhante ao *thingsGateway* do MuFFIN [36, 40]. Desta forma consegue-se interagir de forma transparente com diferentes tipos de redes e dispositivos.

2 Armazenar dados

Desafio: Por vezes é necessário aceder a dados mais antigos para se poder efetuar estatísticas. Atendendo ao facto dos dados não serem enviados diretamente das redes de sensores para as aplicações cliente, há a necessidade de manter a informação recolhida pelas redes de sensores durante um período de tempo pré-definido.

Abordagem a seguir: Recorrer a uma base de dados relacional para armazenar os dados recolhidos pelas redes de sensores. Mantendo uma referência temporal sobre esses dados, permite o seu acesso durante um período de tempo pré-estabelecido, colocando a informação disponível às aplicações cliente. Há ainda a possibilidade

de estes dados não terem validade, contudo, para que o sistema se mantenha eficiente devem ser feitas limpezas à base de dados periodicamente dado que se tratam de dispositivos com capacidades de computação e armazenamento reduzidas.

3 **Disponibilizar os dados**

Desafio: Os dados devem poder ser acedidos de forma fácil e transparente por qualquer tipo de dispositivo.

Abordagem a seguir: Criar um módulo que abstraia o *middleware*. Este módulo terá uma interface bem definida tanto para consulta como para o envio de dados, podendo disponizar as funcionalidades via serviços na *web*.

4 **Permitir a subscrição de serviços**

Desafio: As aplicações devem subscrever serviços para receberem alertas sobre situações que estão interessadas.

Abordagem a seguir: Implementar um módulo que permita às aplicações subscrever serviços e receber notificações de acordo com as preferências especificadas.

5 **Ser executado em sistemas com capacidades de processamento e memória reduzidas**

Existem já várias opções de *middleware* desenvolvido que cumprem os quatro requisitos apresentados acima. Contudo, a maior parte dessas arquiteturas foram criadas para correr em sistemas com elevada capacidade de computação, como é o caso do MuFFIN, que foi desenvolvido em Java EE, adequado a correr em servidores aplicacionais. Mas, uma vez que esta arquitetura deverá ser implementada em sistemas com baixa capacidade de computação, é necessário desenvolver de raiz grande parte do sistema atual, aproveitando ainda ideias de outros projetos relacionados. Desta forma, toma-se um conjunto de decisões em relação à linguagem de programação utilizada, à base de dados, modo de mapeamento dos dados armazenados e forma de interligação entre os componentes de *software* desenvolvidos.

Com a concretização destes objetivos consegue-se obter um *middleware* que atende aos requisitos do sistema que se pretende desenvolver. Além disso, permite ainda integrar redes de sensores heterógeneas de modo a criar novos sistemas e desenvolver novas aplicações para fins variados. Sendo que a vantagem deste *middleware* em relação a outros é poder ser executado em sistemas com capacidades reduzidas em termos de processamento e memória.

1.2.3 Aplicação

A terceira parte do sistema consistirá numa aplicação cliente, que funcionará com dados recolhidos pela rede de sensores, que serão obtidos através dos serviços disponibilizados

pelo *middleware*. O seu objetivo é evitar acidentes relacionados com concentrações de gases tóxicos, temperaturas altas e desgaste físico dos bombeiros em cenários de incêndios florestais.

A aplicação deverá permitir monitorizar a exposição dos bombeiros a gases tóxicos e temperaturas elevadas, alertar para a alta concentração de gases tóxicos e localizar equipas ou bombeiros.

1.2.3.1 Ricos da aplicação

A aplicação irá operar em cenários de incêndios florestais, onde por vezes os meios se encontram bastante longe uns dos outros. Este facto pode trazer problemas ao nível da comunicação entre as aplicações cliente e as estações base instaladas nos veículos de combate a incêndios.

Segundo Chinthaka Dissanayake *et al.* [24], em condições ambientais normais, as ondas rádio seguem a curva da terra devido à pressão atmosférica e temperatura ambiente, que causam um elevado índice de refração. Contudo, em ambientes de incêndios florestais, as altas temperaturas e as alterações nos gases da atmosfera afectam fortemente o índice de refração criando situações de subrefração, fazendo com que as ondas de rádio se afastem da terra. Acrescentam ainda que a dispersão de partículas libertadas nos incêndios contribui para a absorção e redistribuição do sinal, fazendo com que perca força e mude de direção. Estes dois factos podem revelar-se um problema tanto na comunicação entre a aplicação cliente e a estação base bem como na comunicação entre a estação base e os restantes nós da rede de sensores.

1.3 Contribuições

Em termos práticos, o sistema deve poder ser utilizado por equipas de combate a incêndios florestais para diversos fins. A monitorização dos bombeiros pelo chefe de equipa durante o combate aos incêndios representa a finalidade principal para que este foi desenvolvido. Contudo, podemos utilizá-lo sem a componente de aplicação móvel para monitorização, servindo apenas para recolher dados para análise futura.

Por outro lado, ainda que este sistema tenha sido construído para captar dados, e monitorizar bombeiros em incêndios florestais, este pode também ser reutilizado para outras àreas. Por exemplo, monitorização de trabalhores de uma mina, ou atividades que exponham os trabalhadores a gases tóxicos. Para conseguir isto basta apenas desenvolver um novo tipo de dispositivo para captura de dados, alterando os sensores de acordo com as necessidades e a programação do controlador.

O *middleware* desenvolvido é um componente de *software* genérico para redes de sensores, o qual já havia sido criado. Após as alterações feitas ao mesmo, este deverá

poder ser utilizado em equipamentos de computação miniaturizados, servindo a qualquer tipo de redes de sensores, para os mais variados objetivos.

1.4 Estrutura do documento

Este documento organizado como se segue:

- **Capítulo 2 - Trabalho relacionado**

  É apresentado o trabalho já desenvolvido anteriormente que em algum ponto se pode relacionar com o sistema que se pretende desenvolver. São descritas as normas propostas para a rede de sensores e para *middleware* a desenvolver. É apresentado *middleware* já desenvolvido para redes de sensores e com objetivos semelhantes aos que se pretendem atingir neste projeto. Por fim, são descritas sucintamente algumas patentes de outros dispositivos eletrónicos que têm como objetivo garantir a segurança de bombeiros durante o combate a incêndios.

- **Capítulo 3 - Análise**

  Capítulo onde se apresentam os requisitos do sistema a desenvolver e onde se faz um enquadramento teórico, para garantir que o sistema ficará funcional, de acordo com dados técnicos e cientificos. Por outro lado, estabelece um conjunto de padrões de *software* a usar no desenvolvimento do sistema.

- **Capítulo 4 - Desenho e Implementação**

  Descrevem-se os passos de desenvolvimento de acordo com os objetivos estabelecidos, trabalho relacionado, análise padrões de desenvolvimento e dados técnicos. É dado ênfase a pormenores técnicos do desenvolvimento de cada uma das partes do sistema, bem como dificuldades surgidas e soluções encontradas durante a fase de implementação.

- **Capítulo 5 - Avaliação**

  Apresenta-se um conjunto de testes, resultados e sua interpretação para comprovar a utilidade, funcionalidade e robustez do sistema como um todo.

- **Capítulo 6 - Conclusão**

  Apresentam-se os conhecimentos adquiridos, as conclusões, as limitações e as descobertas, tanto a nível pessoal como acerca das tecnologias utilizadas. Neste capítulo são enumeradas as conclusões tiradas após o estudo e análise de um vasto leque de tecnologias, bem como algumas que surgiram consecutivamente aos testes de um sistema inovador, como o que foi desenvolvido neste trabalho.

Capítulo 2

Trabalho relacionado

2.1 Normas para redes de sensores

Nos últimos anos as redes de sensores começaram a estar presentes nas mais variadas áreas, tais como na emergência médica, na monitorização e controlo de poluição do ar, na segurança de pessoas e edifícios, na deteção de catástrofes ambientais, na indústria aeroespacial e até mesmo na área militar. Além disso, começou a haver cada vez mais interesse em interligar redes de sensores com diferentes tecnologias de comunicação.

Com a evolução das redes de sensores, houve a necessidade de uniformizar a a maneira como a informação das mesmas e os dados recolhidos por estas são disponibilizados na *Web*, por forma a tornar mais fácil a comunicação e a integração com aplicações terceiras. Foi com base em necessidades destas que a *Open Geospatial Consortium* (OGC) criou o *Sensor Web Enablement* [17] (SWE). O SWE é uma norma padrão para explorar sensores e sistemas de sensores conectados na *Web*. Esta norma tem como objetivo permitir que diferentes tipos de sensores e atuadores possam ser acedidos e controlados através da *Web*.

A norma SWE é propõe um conjunto de modelos e serviços que permitem implementar redes de sensores orientadas a serviços e aplicações clientes que utilizam essas mesmas redes. A disponibilização da informação recolhida por sensores na *Web* foi proposta através do estabelecimento de esquemas que descrevem rigosamente os sensores e as suas observações, descrição de interfaces bem definidas para *web services*. Desta forma, a norma SWE conta com as seguintes especificações:

- ***Observations & Measurements* (O&M)** Esta especificação segue os conceitos de observação e mediação propostos por Martin Fowler no livro *Data Analysis Patterns: Reusable Object Models* [28].

  Medição é um conceito criado com vista a simplificar a tarefa de lidar com a grande quantidade de dados recolhidos sobre um determinado fenómeno. Por exemplo, se quisermos quantificar a poluição do ambiente em determinados locais podemos definir a entidade "ArAmbiente" com os atributos que pretendetemos quantificar (*e.g.*, CO_2, CO, O_3, NO_2). Cada instância dessa entidade corresponde a uma leitura da

qualidade do ar num determinado local. Caso se pretenda guardar as variações de valores desses parâmetros ao longo de um dia, há que criar um "Fenómeno" que caracteriza os parâmetros do ar que queremos medir. Ao longo do dia são efetuadas várias medições, cada uma delas associada a um valor. Este valor pode ser um atributo da entidade Medição ou uma instancia de outra entidade específica para lidar com a noção de "quantidade", consoante a necessidade e o nível de abstração que se pretenda para lidar com várias unidades de medida. Assim, o "ArAmbiente" passa a estar associado a várias instâncias da entidade "Medição" para cada instância da entidade "Fenómeno". O modelo entidade-associação apresentado na Figura 2.1 descreve o exemplo discutido.

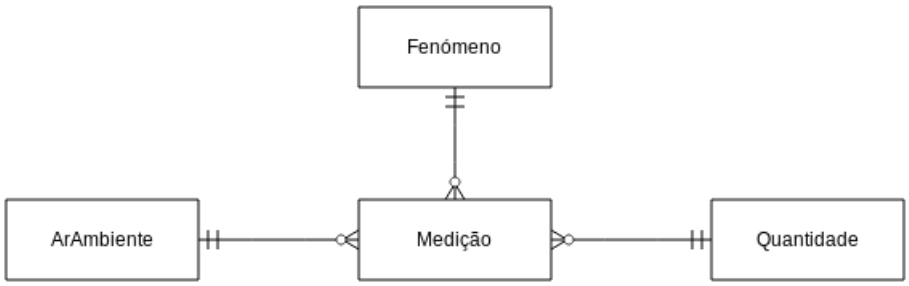


Figura 2.1: Exemplo do uso de medição e tipo de fenómeno para registar a qualidade do ar ambiente

O conceito *Observação* corresponde a um evento que gera um valor, descrevendo um fenómeno. O resultado de uma observação pode ser obtido através de sensores ou de observadores, procedimentos analíticos, simuladores ou outros processos numéricos. Para Martin Fowler [28] este conceito surgiu da necessidade de classificar qualitativamente um objeto. Acrescenta ainda que, por vezes, esta classificação pode ser feita de forma automática inferindo factos com base em observações quantitativas desses mesmos objetos.

Com base nestes dois conceitos (medição e observação) a OGC criou a especificação O&M explicada em [17]. É um modelo para representação e troca de resultados de observações. Permite aceder e trocar observações com diferentes tipos de sensores diminuindo a necessidade de suportar diferentes tipos de dados. Usufrui da flexibilidade e extensibilidade caraterísticas do XML com a capacidade de armazenar grandes quantidades de dados, tanto de texto como blocos binários.

- ***Sensor Model Language* (SensorML) [4]**

  Esta norma fornece modelos XML para descrever processos, que podem ser medições efetuadas por sensores ou instruções para derivar informação a partir de ob-

servações. Em cada um dos processos são definidos as entradas, saídas, procedimentos para tratar os dados e metadados para descoberta das redes de sensores.

A SensorML tem a finalidade de:

- Disponibilizar a especificação de componentes e sistemas de sensores;
- Permitir a descoberta dos sistemas de sensores e dos processos de observação através de metadados;
- Descrever cadeias de processos de geolocalização ou processos de observações disponíveis através da *Web*, e executá-los sem conhecimento *à priori* das suas caraterísticas;
- Dar suporte aos serviços SOS, SPS e SAS (explicados individualmente nas secções seguintes);
- Desenvolver sensores *plug-n-play*, simuladores, e processos de forma a serem integrados em sistema de apoio à decisão.

- ***Sensor Observations Service* (SOS) [17]**

  O SOS é um padrão para permitir o acesso a observações de sensores e sistemas de sensores de forma uniforme, ultrapassando o problemas de heterogeniedade das redes de sensores. Consiste numa API para gestão dos sensores implementados e receção dos dados dos sensores, mais especificamente dos dados das observações.

  Este é um elemento essencial para o SWE, uma vez que define a representação dos dados na rede e as operações para aceder e integrar dados de observações a partir de redes de sensores. Posiciona-se como um intermediário entre um cliente e um repositório de observações.

- ***Sensor Planning Service* (SPS) [11]**

  É uma especificação que define *interfaces* para obter informações acerca das capacidades e caraterísticas de um sensor. Oferece operações para:

  - Determinar a viabilidade de pedidos a sensores;
  - Determinar o estado de tais pedidos;
  - Atualizar ou cancelar os pedidos feitos;
  - Pedir informações acerca de outros *web services* que fornecem acesso aos dados guardados.

  Em suma, é uma interface para um serviço na *Web* que permite obter a viabilidade funcional das redes de sensores.

- ***Sensor Alert Service* (SAS) [17]**

  O SAS é um sistema de notificação de eventos que interpreta cada nó de uma rede de sensores como um objeto de interesse. Cada nó anuncia as publicações que são recebidas pelos interessados (aqueles que subscrevem o serviço). Ou seja, quando um sensor tem uma nova observação, este anuncia o valor que desencadeia um processo de envio de um alerta aos nós da rede interessados nessa observação. A informação dos subscritores é mantida numa tabela de subscrições local a cada nó.

  Esta norma usa o protocolo *eXtensible Messaging and Presence Protocol* (XMPP), que este dispõe da funcionalidade *push-based notification*.

- ***Web Notification Services* (WNS) [17]**

  Esta especificação fornece uma *interface* que permite aos clientes estabelecerem comunicação assíncrona com serviços. Efetua o encaminhamento de mensagens entre clientes e serviços ou entre serviços, converterendo mensagens, quando não usam o mesmo protocolo. Note-se que o WNS não se comporta como um sistema de alertas (*cf.* SAS), mas pode ser usado caso os clientes pretendam receber as notificações de forma assíncrona, por exemplo por SMS. Desta forma, mantêm-se conexos permanentemente à espera de resposta.

No sistema foram utilizados alguns dos padrões descritos anteriormente, os quais serão descritos de seguida.

O padrão O&M foi utilizado para estruturar os dados recebidos das leituras feitas aos gases e temperatura durante os incêndios florestais. Ou seja, foram estruturadas observações baseadas nas medições feitas, ao longo do tempo de permanência no incêndio, aos fenómenos pré-definidos (monóxido de carbono, dioxido de azoto e tempeartura).

O SOS foi um dos principais padrões utilizados no desenvolvimento do projeto. Isto porque foi utilizado para uniformizar os dados recebidos das redes de sensores pelo *middleware*facilitando, posteriormente, a utilização desses mesmos dados por terceiros. Em suma, permite que qualquer cliente aceda às observações armazenadas e que estas sejam de fácil entendimento.

O SAS foi implementado no *middleware*, embora não seja utilizado pela aplicação cliente. Desta forma consegue-se garantir que posteriormente podem ser adicionadas funcinalidades ao sistema que necessitem de receber alertas emitidos pelos serviços do *middleware*.

2.2 *Middleware*

Existem vários sistemas *middleware* focados na "Internet das Coisas", que garantam comunicação com redes de sensores independentemente das tecnologias usadas. De seguida apresento alguns desses sistemas:

- **GSN *Middleware* [14, 15]:**

  Desenvolvido no âmbito do projeto *Global Sensor Network* (GSN) [16] na *École Polytétchnique de Lausanne*. Tem como objetivo fornecer abstrações e meios flexíveis para integrar plataformas físicas (*e.g.*, sensores), sendo os sensores virtuais a chave para a sua abstração. Este *middleware* acelera o processo de implementação de redes de sensores e ajuda a criar APIs que simplificam o desenvolvimento de aplicações, disponibilizando as suas funcionalidades na forma de serviços na *Web*.

  Para o GSN, sensores virtuais são entidades produtoras de dados (*e.g.*, uma câmara de filmar, um sensor de temperatura, um computador, ou até mesmo uma combinação de sensores virtuais), podendo ter vários fluxos de entrada de dados e um fluxo único de saída. Estes sensores virtuais são usados para abstrair os detalhes de implementação para acesso aos dados, possuindo uma especificação para a sua implementação e uso. Essa especificação é feita em ficheiros *Tranducer Electronic Data Sheet* (TEDS) especificados segundo a norma IEEE-1451 [30].

  A plataforma permite a implementação e integração rápida de redes de sensores heterogéneas. Caracteriza-se pela simplicidade, adaptabilidade, escalabilidade e fácil implementação. A simplicidade resulta das abstrações que propões, permitindo modelar de forma declarativa as redes de sensores e os dados, usando XML e SQL para a sua manuipulação. A adaptabilidade deve-se ao facto de permitir adicionar novos tipos de redes de sensores e facilitar a sua reconfiguração em tempo de execução. Suporta um grande número de produtores e consumidores de dados com requisitos aplicacionais muito distintos, graças à sua escalabilidade. Usufrui da arquitetura *peer-to-peer* para distribuir as consultas de dados e descoberta de redes de sensores. A utilização de sensores virtuais, descritos através de ficheiros XML, contendo propriedades caraterísticas de medida do sensor, torna mais fácil a sua implementação. Esta descrição evita a programação em baixo nível das características dos dispositivos. Há ainda que referir que o GSN é facilmente implementável num ambiente padronizado de computação sem necessitar de grandes requisitos de *hardware*, e é desenvolvido em Java o que garante portabilidade.

  Em termos de adaptabilidade, o *middleware* desenvolvido permite, como o GSN, a instalação e reconfiguração de novos serviços em tempo de execução. A nível da escalabilidade, apesar do *hardware* ter limitações a nível de processamento para atender a um grande número de produtores e consumidores, o *software* desenvolvido tem capacidade para suportar um aumento de carga, independentemente das tecnologias de comunicação utilizadas. Garante ainda uma portabilidade semelhante à do GSN, pelo facto de ter sido desenvolvido em Java.

- **52 *North - Sensor Bus* [18]**

  A comunidade 52 *North* tem como principal objetivo o desenvolvimento de projetos relacionados com os padrões SWE da OGC (ver secção anterior). Assim, têm projetos de *middleware* que suportam padrões como o SOS, o SAS ou o SPS, tal como descrito no *Sensor Bus* [18].

  O *Sensor Bus* posiciona-se como uma camada intermédia entre as redes de sensores e as aplicações cliente, facilitando a utilização de sensores em aplicações cliente. Esta arquitetura pode suportar diferentes tipos de comunicação de acordo com os adaptadores implementados para a comunicação com as redes de sensores, uma vez que esta arquitetura disponibiliza uma interface bem-definida para implementá-los. O *Sensor Bus* permite também implementar novos tipos de serviços para aplicações cliente. Quanto ao modo de funcionamento, mantém sempre associações entre os serviços e as *Sensor Gateways* que permitem aceder aos sensores que dão resposta a esses serviços.

  Além de interfaces bem-definidas para permitir a sua extensibilidade, o *Sensor Bus* conta com uma infraestrutura de comunicação comum e um protocolo de mensagens. A infraestrutura de comunicação é baseada no mecanismo *publish/subscribe*. O protocolo de mensagens é garantido com a ajuda dos seus *gateways* que convertem o protocolo de comunicação específico do dispositivo no protocolo de comunicação interno ao *middleware* e vice-versa.

- **MuFFIN [36, 40]**

  O MuFFIN é um *middleware* desenvolvido em JavaEE e corre no servidor aplicacional *Fuse Enterprise Service Bus* (FuseESB), que tem elevados requisitos de *hardware*, principalmente ao nível da memória RAM (necessita de 2GB de memória). Quanto à persistência dos dados o MuFFIN usa uma base de dados MySQL, e JPA como plataforma de mapeamento de dados entre o modelo relacional e o modelo orientado a objetos.

  O MuFFIN implementa alguns dos padrões da OGC para redes de sensores, como por exemplo o SensorML ou SOS o (ver secção 2.1). A implementação destes padrões permite a integração de várias redes de sensores heterogéneas independentemente do tipo de sensores utilizados. Ainda quanto à caraterística de extensibilidade, o MuFFIN usufrui de um componente (o *ThingsGateway*) que permite integrar redes de sensores heterogéneas, podendo estas usar diferentes tipos de comunicação com diferentes tipos de redes de sensores.

  Ao nível da implementação das normas da OGC, este *middleware* consegue, por exemplo, comunicar com sensores de vários fabricantes facilmente, uma vez que as informações necessárias para a tradução dos dados lidos pelo sensor estão explícitos

em ficheiros XML (SensorML).

Em termos de suporte a vários tipos de comunicação, o *ThingsGateway* possui uma interface bem definida para adicionar facilmente ao mesmo novas tecnologias de comunicação. Assim, cada vez que for necessários agregar um nova rede de sensores, com uma tecnologia de comunicação das já existentes, basta adicionar um *gateway* implemente essa *interface* disponibilizada de acordo com as características da rede de sensores e da tecnologia de comunicação pretendida.

2.3 Projetos aplicados ao trabalho dos bombeiros

Um ponto essencial da pesquisa sobre trabalho relacionado foi encontrar projetos semelhantes ao que se propõe, ou seja, com intuito de dar apoio ao trabalho dos bombeiros, independetemente da área específica. Tanto o resultado das pesquisas efetuadas, como pela minha experiência pessoal acerca do trabalho dos bombeiros, na área dos incêndios florestais tem havido poucos progressos em termos de equipamentos de segurança e apoio aos bombeiros no teatro de operações, quando comparado com a área dos incêndios urbanos e industriais. Assim, serão apresentadas algumas patentes de projetos com objetivos idênticos aos do sistema que se pretende desenvolver, mas aplicados a outras áreas de intervenção dos bombeiros que não a incêndios florestais.

- **Sistema de medição de gás [39]**

  Patente americana para um sistema de medição da concentração de gases, composto por um dispositivo móvel para medição de gases e uma estação base para carregamento do dispositivo de medição. Uma das principais finalidades deste sistema é dar auxílio a bombeiros no combate a incêndios urbanos e industriais ou em acidentes com matérias perigosas, cenários onde existe uma enorme variedade de gases tóxicos, e por vezes exposições de curta duração a esses gases pode causar Doença Pulmunar Obstrutiva Crónica (DPOC) podendo mesmo levar à morte. Este dispositivo é essencial para determinar o tipo de perigos a que se está exposto e quais as medidas de segurança que devem ser adotadas no teatro das operações, em termos de equipamentos de proteção individual e tempos de exposição a estes gases. O dispositivo de medição pode conter um ou mais sensores de gás dependendo do fim a que se destina. Ao juntar o dispositivo de medição à base é possivel carregar as suas baterias e descarregar os dados recolhidos.

  Contudo, este dispositivo não satisfaz os requisitos do dispositivo pretendido, pois não permite obter a localização do bombeiro que o transporta nem dispõe de mecanismos de comunicação sem fios para envio dos dados recolhidos para uma estação base ou dispositivo de monitorização das equipas que os utilizam.

- **Sistema de gestão e controlo de bombeiros em incêndios urbanos e industriais [29]**

  Este sistema implementa algumas das funcionalidades que se prentedem, tais como a deteção da temperatura a que os bombeiros estão expostos e a capacidade de monitorizar um conjunto de bombeiros em trabalho. Contudo, é aplicado à área do combate a incêndios urbanos e industriais, e é limitado em certos aspetos, como por exemplo, (1) a extensibilidade do sistema, uma vez que cada painel de controlo pode monitorizar até doze bombeiros, e não há indicações que este possa ser expandido para controlar mais elementos; (2) adicionar novas aplicações ou funcionalidades ao painel de controlo; ou (3) comunicar com outros tipos de dispositivos.

  Em suma, este sistema é composto por dispositivos móveis que acompanham os bombeiros no teatro de operações e por uma estação base (painel de controlo) que deve permanecer fora do edifício onde se está a atuar. A estação deve permanecer sempre junto do *Entry Control Officer* - indivíduo responsável por registar e controlar todos os bombeiros que entram dentro do edifício em chamas, bem como o tempo que cada um permanece lá dentro - para que as suas funções sejam executadas mais eficientemente e com maior precisão e rapidez. Os dispositivos móveis são capazes de monitorizar os parâmetros críticos dos homens dentro do edifício, tais como:

  - Pressão do Aparelho Respiratório Isolante de Circuito Aberto (ARICA) - Sendo o ARICA o equipamento que permite aos bombeiros respirar com segurança dentro dos edifícios em chamas, ou em cenários com gases tóxicos, é necessário determinar qual é a capacidade restante das garrafas, de forma a poder-se estimar qual o tempo que cada elemento ainda pode permanecer dentro do edifício;
  - Tempo para o alarme - O alarme é um sistema de segurança dos ARICAs que normalmente dispara quando a capacidade restante é de 50 bar. Segundo o esforço de um bombeiro em trabalho possibilita em média cinco minutos de ar respirável no aparelho. Desta forma o alarme emite um sinal sonoro para indicar ao bombeiro que tem que abandonar o edifício.
  - Temperatura ambiente - Uma vez que as temperaturas a que os bombeiros estão expostos em cenários de incêndios urbanos são muitos altas é necessário monitorizar essas temperaturas para proteção dos indivíduos.
  - Tempo desde que o bombeiro está a usar o ARICA.

  Este dispositivo contém ainda uma porta para carregamento da bateria e para descarregar dados. Quanto ao painel de controlo, comunica com os dispositivos móveis

de forma a obter a informações de monitorização dos bombeiros, e mostra as informações, possibilitando ainda carregar as baterias dos dispositivos móveis.

Este sistema é semelhante o que se prentende desenvolver. Contudo, a finalidade é bastante diferente. Os dispositivos que acompanham os bombeiros durante o incêndios, em vez de medirem os tempos de permanecia no edfifício a pressão do ARICA devem medir concentrações de gases e obter as coordenadas da localização dos mesmos. Quanto à medição da temperatura, esta deve manter-se. Matém ainda semelhanças ao nível do painel de controlo. No entanto, o sistema que se pretende não deve ser limitado ao número de bombeiros que se monitorizam. Deve ainda algo que seja facilmente transportado pelos chefes de equipa durante os incêndios, surgindo então a ideia de uma aplicação para *smartphone*.

Capítulo 3

Análise

Este projeto tem como grande desafio a construção de um sistema de *software* suficientemente robusto e estruturado, capaz de tratar e armazenar a informação recolhida pelos dispositivos móveis e disponibilizá-la a aplicações cliente de forma organizada. Cumulativamente a este requisito, acrescenta-se a restrição da utilização de equipamentos de computação miniturizados, com baixo consumo energético, fácil instalação em viaturas e baixo custo.

Por outro lado, ao falar-se sobre recolha de dados em ambientes hostis, mais precisamente em cenários de temperaturas altas, surgem de imediato várias questões, a saber: (1) como criar um isolamento dos dispositivos de recolha de dados às altas temperaturas a que possam estar expostos; (2) como garantir que o isolamento dos dispositivos não afeta fortemente a comunicação para envio dos dados recolhidos; (3) como evitar a perda total de dados devido à ação do calor nas ondas de comunicação sem fios; garantir uma distância minima de comunicação entre os dispositivos para envio dos dados recolhidos.

Por forma a resolver estas questões foram feitas pesquisas que o documento de seguida apresenta, bem como alguns testes que serão apresentados no Capítulo 5.

3.1 Dispositivos de computação miniaturizados

De modo a preencher os requisitos de portabilidade do sistema e ter equipamentos de baixo consumo energético, é necessário que a estação base de armazenamento dos dados recolhidos seja desenvolvida sobre um dispositivo de computação miniaturizado.

Existem algumas soluções no mercado que satisfazem os nossos requisitos. Neste projeto, optámos como dispositivo a ser usado para estação base o Raspberry Pi. Esta escolha deveu-se principalmente ao seu mediatismo, pois é um dispositivo muito utilizado hoje em dia, e sobre o qual existe muita informção disponível, permitindo assim que seja mais fácil a sua programação e configuração, e procura de soluções para algum problema que surja.

3.1.1 Raspberry Pi

O Raspberry Pi é um computador do tamanho de um cartão de multibanco, contendo todo o *hardware* numa única placa de circuito impresso. Esta pode ser facilmente expandida integrando outro tipo de *hardware* consoante as necessidades do utilizador. Mais especificamente, estes dispositivos contam com um processador da família ARM, uma unidade de processamento gráfico, memória SDRAM e uma *slot* para cartões SD, que serve para armazenar informação de forma persistente. Existem atualmente três modelos diferindo apenas nalgumas caraterísticas que se apresentam na tabela seguinte.

Modelo	A	B	B+
Número de portas USB	1	2	4
Memória RAM	256MB	512MB	512MB
Ethernet (porta RJ45)	Não	Sim	Sim
Número de pinos GPIO	26	26	40

Tabela 3.1: Tabela de comparação de modelos Raspberry Pi

Dadas as caraterísticas de todos os modelos e a baixa diferença de custos entre eles, optou-se pelo modelo B, uma vez que pode trazer vantagens ao nível da memória RAM.

3.1.2 Sistema operativo

Optou-se pelo Raspbian [10] baseado nas recomendações disponíveis no sítio do Raspberry Pi, na página dedicada ao *software* [9]. A recomendação feita neste sítio deve-se ao facto deste sistema operativo ter sido desenvolvido pela mesma equipa que criou o Raspberry Pi, e ter diversas vantagens, tais como, o facto de ter sido desenhado especificamente para estes dispositivos, e por estar em constante atualização, bem como por desde logo incluir ambientes de progrmação em linguagens como o Python [7] e o JavaSE [6].

3.2 Middleware para dispositivos de computação miniaturizados

Como referido no capítulo anterior, o *middleware* MuFFIN é capaz de preencher alguns dos objetivos pretendidos. Em contrapartida, este sistema não está preparado para ser executado em dispositivos computacionais miniaturizados, principalmente devido às bibliotecas e componentes que usa. Desta forma, pretende-se reconstruir o MuFFIN, mantendo as suas funcionalidades, mas retirando alguns desses componentes que o tornam demasiado "pesado" para o dispositvo alvo. Uma vez que o MuFFIN foi desenvolvido em Java, esta variante do mesmo será também desenvolvida em Java de modo a reaproveitar alguns dos componentes que não necessitem de ser reprogramados.

Para poder organizar a reconstrução do MuFFIN, foram analisados todos os componentes para detetar aqueles que possam pôr em causa a eficiência do sistema. Os pontos levantados foram:

- Alguns componentes internos do MuFFIN, podem causar um processamento lento ou consumir demasiada memória, tais como o uso da biblioteca JPA Hibernate que implementa e o uso de SOAP ao invés de REST para disponibilizar serviços na *Web*;
- Quanto a componentes externos ao MuFFIN verificámos o uso do servidor aplicacional FuseESB para integração e troca de mensagens entre todos os componentes do MuFFIN.

Foi decidido reconstruir a camada de persistência de dados com a finalidade de retirar o JPA, bem como criar outra forma de integração e comunicação entre os compoentes do sistema. Quanto aos serviços na *Web* foi mantido o protocolo SOAP, uma vez que é a melhor forma de organizar e estruturar os pedidos e respostas dos serviços, principalmente quando temos pedidos que podem ter inúmeras restrições e filtros baseados em SWE.

3.2.1 Camada de acesso aos dados

Para efetuar o mapeamento entre o modelo de objetos e o modelo relacional experimentou-se os seguintes padrões, baseados em [27]:

- *Active Record* - Segundo este padrão, cada objeto corresponde a um registo de uma tabela da base dados, em que os atributos correspondem às colunas da tabela. Além disso, cada objeto deve possuir métodos correspondentes às acções de inserção, remoção, atualização e pesquisa de entradas da base dados.

  Por exemplo, cada entrada na tabela Sensor representada na Figura 3.1 corresponde a uma instância do objeto descrito na Lista 3.1.

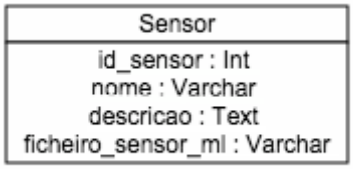


Figura 3.1: Tabela Sensor

Lista 3.1: Exemplo de objeto que segue o padrão *Active Record*

```
public class Sensor {
        private String nome;
        private String descricao;
        private String ficheiroSensorML;

        private static final String
                findStatement = "...";
        private static final String
                updateStatement = "...";
        private static final String
                insertStatement = "...";

        public static Sensor find(Long id){
                ...
        }

        public static void update(){
                ...
        }

        public static Long insert(){
                ...
        }

        public String getNome(){
                ...
        }

        public String getDescricao(){
                ...
        }

        public String getFicheiroSensorML(){
                ...
        }

}
```

- *Data Mapper* - Este padrão dita que deve existir uma camada que mapeia a camada de dados e a representação de objetos do domínio em memória. Um dos objetivos deste padrão é isolar a representação dos dados em mémoria da base de dados.

  Tal como no padrão *Active Record* devem existir objetos específicos para cada tabela, onde cada instância representa registos da tabela da base de dados, e os atributos correspondentes às colunas da tabela. Por outro lado, estes objetos não possuem

métodos para as operações na base de dados, ficando estas operações a cargo da camada intermédia de mapeamento. Esta camada intermédia é constituída por um ou mais *mappers* que efetuam as operações de inserção, remoção, leitura e atualização na base de dados.

Após análise de ambos os padrões acima descritos optou-se pelo *Data Mapper* ao invés do *Active Record*.

De modo a tratar a relação entre objetos, os padrões de desenho analisados foram:

- *Indentity Map* - Padrão de desenho que garante que cada objeto que representa registos da base de dados só existe em memória uma única vez, fazendo uso de uma estrutura de dados do tipo *Map* para guardar referências dos objetos.

  A necessidade de só ter um objeto em memória por cada registo prende-se com o facto de garantir que as atualizações de informação à base de dados ocorre de forma correta. Se existirem dois objetos em memória para o mesmo registo na base de dados, se estes forem atualizados de forma concorrente , isto pode provocar conflitos na gravação de informação para a base de dados.

  Neste modelo, uma vez que usamos uma estrutura do tipo *Map* sabemos à partida que iremos necessitar de uma chave para aceder e referir-nos a cada um dos objetos. Tipicamente é usada como chave da estrutura de dados a chave primária correspondente na tabela da base de dados, pois devem sempre ser usados tipos de dados simples como por exemplo inteiros ou *strings*.

- *Lazy Load* - Frequentemente utilizado com intuito de poupar memória bem como de baixar o tempo de processamento quando existem objetos com muitas referências para outros objetos, evitando assim que se perca tempo com inúmeros acessos ao disco a carregar esses mesmos objetos a partir da base de dados.

  Em termos de desenho este padrão apresenta-se como uma camada de abstração de um objeto, contendo os *getters* e *setters* do objeto pretendido sem guardar em memória os valores dos seus atributos. Tipicamente este padrão recorre também ao uso do padrão *Factory Pattern Method* e *Singleton Pattern* (secção 3.2.1.1), por forma a manter uma referência para a implementação do objeto com os atributos após este ser criado.

Lista 3.2: Exemplo de implementação do padrão *Lazy Load* em Java

```
public class SensorProxy extends Proxy<Sensor>
                                    implements Sensor {

    protected Sensor object;

    public String getNome(){
```

```
            return createObject().getNome();
        }

        public void setNome(String nome){
            createObject().setNome(nome);
        }

        //restantes getters e setters

        protected Sensor createObject(){
            if(object == null){
                //cria novo sensor atraves
                //de Objecto especifico para
                //tal por exemplo: uso de
                //um mapper
            }
            return this.object;
        }
}
```

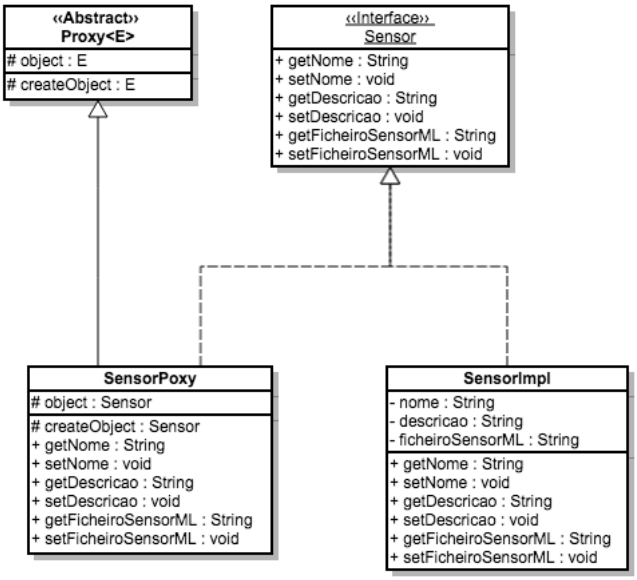


Figura 3.2: Exemplo de *Lazy Load*

Como podemos observar no diagrama da Figura 3.2, tanto a implementação da classe Sensor como a da classe de *Lazy Loading* correspondente têm a mesma interface "Sensor". Assim, os objetos têm o mesmo supertipo, o que permite que sejam usados permutavelmente e quando chamamos um método de um objeto não sabemos se estamos realmente a aceder a uma instância de *Lazy Loading* ou à implementação concreta do objeto.

Quanto à implementação do *Lazy Load*, este limita-se a delegar as chamdas para a instância da implementação concreta do tipo "Sensor" ("SensorImpl"). Desta forma, os seus dados só serão carregados para memória quando necessitarmos de aceder a algum atributo da instância que estamos a utilizar, caso contrário este objeto permanecerá em memória fazendo referência a um sensor sem guardar os seus atributos.

Por último, para lidar com problemas de chaves estrangeiras, associações de um-para-muitos e associações de muitos-para-muitos foram implementados os seguintes padrões:

- *Foreign Key Mapping* - Num sistema orientado a objetos, em que um objeto refere outro é necessário refletir esta ligação de forma persistente na base de dados. Contudo, não é viável guardar a referência dos objetos na base de dados, uma vez que quando iniciarmos de novo sistema e os objetos forem carregados em memória ser-lhe-ão atribuídas novas referências. Desta forma, surge a necessidade de mapear um objeto através de chaves estrangeiras, geralmente por meio de um tipo de dados primitivo (*e.g.*, inteiros) na base de dados, podendo mapear de forma mais ou menos complicada as associações entre objetos no modelo relacional.

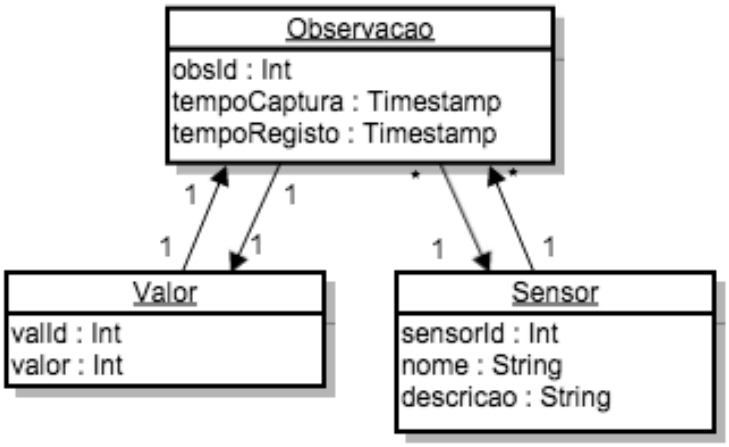


Figura 3.3: Exemplo de mapeamento de chave estrangeira segundo modelo de objetos

As Figuras 3.3 e 3.4 mostram um sistema com a classe "Observacao", em que cada instância referencia um "Valor" e vice-versa, bem como cada "Sensor" referencia uma ou várias instâncias do tipo "Observacao". A forma correta de mapear estes objetos é conforme as tabelas apresentadas na Figura 3.4, ou seja, uma vez que a cada "Observacao" corresponde um "Valor", na tabela "valor" existe um atributo referente ao identificador da "Observacao" a que cada "Valor" (cada entrada na tabela "valor") corresponde. Por sua vez, cada nova entrada na tabela "observacao" referencia uma chave primária da tabela "sensor". Desta forma, as referências

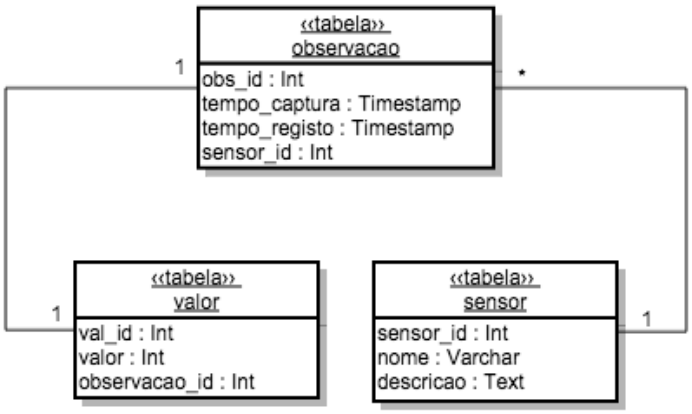


Figura 3.4: Exemplo de mapeamento de chave estrangeira segundo modelo relacional

no modelo orientado a objetos são substituídas por campos que contêm as chaves primários dos elementos que referenciam.

- *Association Table Mapping* - Segundo a primeira forma normal [19] não podem existir múltiplos valores num único campo na base de dados. Portanto, num cenário onde há um conjunto de objetos que podem referenciar diretamente um conjunto de outros objetos (e vice-versa), como o exemplo apresentado na Figura 3.5, cada "Observacao" pode estar ligada a vários objetos do tipo "Grupo", e cada "Grupo" pode referenciar vários objetos do tipo "Observacao", há necessidade de efetuar um mapeamento através de uma tabela de associação.

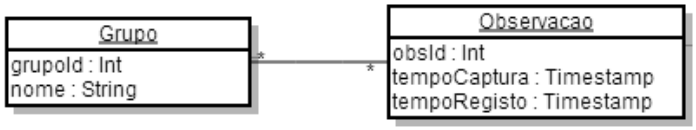


Figura 3.5: Exemplo de associação de muitos-para-muitos (modelo de objetos)

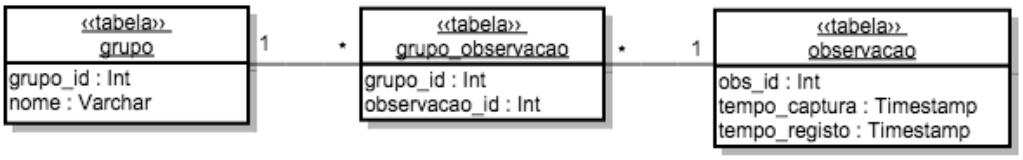


Figura 3.6: Exemplo de associação de muitos-para-muitos (modelo relacional)

Este mapeamento consiste na criação de uma tabela adicional com dois campos constituídos por chaves estrangeiras, formando as associações com base nas chaves primárias das tabelas pretendidas. Desta forma é possivel criar uma associação

de muitos-para-muitos entre duas tabelas na base de dados, como podemos ver na Figura 3.6.

3.2.1.1 Padrões e métodos de desenho auxiliares

Para desenvolver os componentes de *software* necessários às alterações pretendidas para o MuFFIN, foi necessário recorrer a alguns padrões de desenvolvimento auxiliares e mais genéricos que os descritos até agora. Contamos então com:

- *Singleton Pattern* - Garante a existência de apenas uma instância de uma classe. Mantém um ponto global de acesso ao objeto para que este seja criado apenas uma vez e possa ser acedido por vários outros objectos. O seu funcionamento pode ser observado na Lista 3.2, onde é utilizado numa *Proxy* de um objeto, de modo a que esta crie apena uma vez o objeto, independentemente do número de vezes que se pretender aceder ao mesmo. Este padrão foi utilizado várias vezes ao longo do projeto, principalmente na implementação do *Lazy Load* em conjunto com o *Factory Method*. Outra aplicação deste padrão foi na criação dos varios repositórios (estruturas de dados que guardam os objetos que estão a ser utilizados, para evitar que se esteja constantemente a aceder ao disco), os quais só são criados à medida que forem necessários e dispõem de um ponto global de acesso, o "RepositoryFactory".

- *Factory Pattern Method* - Permite adiar a criação de objetos para as subclasses, deixando estas decidir como criar o mesmo. Este método pode assumir duas variantes que são: (1) o caso em que a classe criadora é uma classe abstrata onde não é definido o método de criação, obrigando a existência subclasses para definir o método de criação que não permitem uma definição por omissão sob pena de instanciar classes imprevisíveis; (2) o caso em que o criador é uma classe concreta e fornece uma definição por omissão para o método de criação. Por exemplo, ao utilizarmos *proxies* de objetos quando aplicamos o padrão *LazyLoad*, podemos recorrer aeste padrão. Em vez de definirmos uma na classe "Proxy" qual o objeto a criar quando foir necessário aceder aos atributos do mesmo, esta responsabilidade fica a cargo da implementção especifíca da *Proxy* para o objeto em causa (*c.f.* Figura 3.2).

- *Template Method* - É um método de desenvolvimento que auxilia na definição de um algoritmo, objeto ou tipo de dados genérico. Possibilita definir alguns métodos de uma classe e deixar outros abstratos, os quais deverão ser definidos pelas subclasses que a implementam. Desta forma, é possível criar uma classe que implementa à partida todas as funcionalidades que serão obrigatoriamente iguais em todas as suas subclasses, e deixar a discrição das mesmas a forma como estas implementam o resto dos métodos de acordo com a sua finalidade.

3.2.2 Integração de componentes de *software*

Tratando-se de um sistema de *software* ao qual podem ser adicionados novos componentes, tais como *gateways* para comunicação com dispositivos externos e serviços para tratamento dos dados recebidos, entre outros, torna-se necessário criar uma infraestrtura de comunicação entre esses componentes que seja eficiente em termos de memória e capacidade computacional. O objetivo é permitir o fluxo de informação entre esses *gateways* e serviços de acordo com os as dependências pré-estabelecidas pelo utilizador, poupando recusos do sistema relativamente ao componente utilizado pelo MuFFIN para esse fim.

Colocam-se duas hipóteses de infraestruturas para preencher esta funcionalidade do sistema. A primeira, consiste em desenvolver uma infraestrtura que crie vários fluxos de informação através de *pipes* de acordo com as dependências pré-estabelecias. A segunda é inspirada nos sistemas baseados em eventos, em que a informação circula encapsulada dentro de eventos. Cada serviço regista-se numa de eventos e subscreve os serviços de onde lhe interessa receber informação de acordo com as dependências acima referidas. Optou-se pela segunda alternativa pois não necessita da criação de canais de comunicação (*pipes*) para cada dependência entre os serviços, poupando desta forma recursos do sistema.

3.3 Dispositivos móveis de captura de dados

3.3.1 Plataforma de prototipagem

Numa fase inicial do projeto utilizámos plataformas de prototipagem de *hardware* livre Arduino [1]. Esta conta com um controlador Atmel AVR, tendo suporte para entrada e saída de dados de forma a facilitar a programação e prototipagem. Em termos de programação, oferece uma linguagem baseada em C/C++. O objetivo destes dispositivos é a criação de protótipos de *hardware* de baixo custo, permitindo a sua expansão com os componentes que se pretender e programá-los. Podem ser usados de forma independente ou ligado a um computador por meio de um cabo USB.

O primeiro modelo destes dispositivos foi lançado em 2005, existindo atualmente perto de 20 modelos com diferentes caraterísticas em termos de processador, memórias SRAM e EPROM, voltagem usada, e a capacidade de entrada e saída de dados. Optámos pelo Arduino *Uno* uma vez que compreende uma das melhores relações qualidade preço. A versão R3 que escolhemos, conta com:

- Processador/Microcontrolador AtMega328 (velocidade de processamento máxima de 16MHz);
- Voltagem de entrada limitada ao intervalo 6-20V;
- 6 pinos de entrada analógica;

- 14 pinos de entrada/saída digital;
- Memória Flash de 32KB;
- Memória SRAM de 2KB;
- Memória EPROM de 1KB;

3.3.2 Isolamento térmico

A maior parte dos equipamentos eletrónicos são sensíveis a altas temperaturas (tipicamente mais de 50ºC). Para a utilização que pretendemos há que criar um isolamento para os dispositivos que recolhem os dados do ambiente, pois vão estar sujeitos a temperaturas altas quando usados em cenários de incêndios florestais. Além disso é importante garantir que este revestimento não aumenta em demasia o peso do dispositivo nem diminui a capacidade de leitura dos sensores e alcance do equipamento de comunicação sem fios.

A solução a que chegámos consiste numa simples caixa de plástico para fechar e proteger os componentes, revestida com um isolante térmico. Contudo, a escolha de um isolante implica a compreensão da forma como se faz a propagação de calor. Existem então três formas de propagação de calor:

- **Convecção** - Forma de propagação térmica através da matéria em movimento devido às diferentes densidades dessa matéria. Este modo de propagação só ocorre em fluídos (*e.g.*, líquidos, vapor e gases). Na Figura 3.7 a chama aquece a água junto ao ponto A, originando a expansão das particulas; esta expansão faz diminuir a sua densidade passando esta a ser mais leve que a água que se encontra no resto do tubo. Desta forma a água mais densa vai descer ocupando o lugar da água quente que ocupa o segmento A-B, empurrando a água menos densa para o segmento B-C. Posteriormente, junto ao ponto A volta a estar água fria, que irá aquecer e movimentar-se como aconteceu anteriormente. A estes movimentos chamamos correntes de convecção, e são fenómenos que ocorrem frequentemente em cenários de incêndios florestais com as deslocações do ar quente.

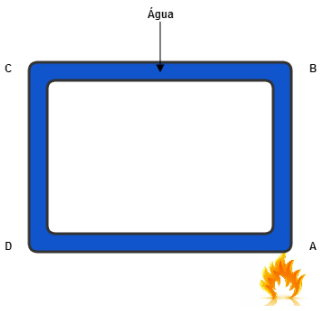


Figura 3.7: Corrente de convecção

- **Condução**

  Ao contrário da convecção, na condução a propagação de calor ocorre sem a deslocação de partículas. Esta forma de propagação térmica é feita através da vibração das moléculas do corpo que está a propagar o calor. Ou seja, a região de um corpo mais próxima de uma fonte de calor sofre o aumento da vibração das suas moléculas. Consecutivamente aumenta a energia cinética das mesmas, sendo o calor transmitido através de choques com as moléculas vizinhas aumentando também a sua vibração.

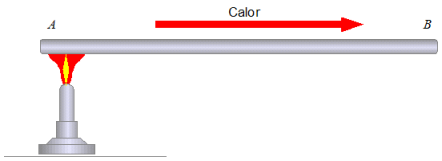


Figura 3.8: Condução térmica
Fonte: http://interferenciafisica.blogspot.pt/2010/03/transmissao-de-calor.html

Os metais são exemplos de bons condutores térmicos, uma vez que possuem os electrões mais externos fracamente ligados, permitindo o transporte de energia por meio de colisões. Em contrapartida, materiais com os electrões exteriores fortemente ligados não possuem tão boa condutividade térmica, como, por exemplo, o vidro, a borracha ou a lã de vidro.

- **Radiação**

  Este tipo de propagação térmica é efetuada através de ondas eletromagnéticas, podendo possuir diversas frequências. O exemplo mais prático de propagação térmica por radiação é a emissão de energia solar, que é propagada através do vácuo até à Terra. Esta energia é composta por vários tipo de ondas (*e.g.*, raios X, raios ultravioleta, luz visível, raios infravermelhos, microondas).

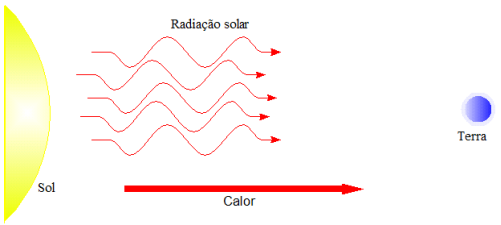


Figura 3.9: Radiação térmica
Fonte: http://interferenciafisica.blogspot.pt/2010/03/transmissao-de-calor.html

Qualquer corpo com temperatura superior ao zero absoluto (0K ou -273ºC) emite energia radiante. As ondas infravermelhas são as que mais facilmente se transformam em calor quando absorvidas pelo recetor, sendo por isso tipicamente chamadas de ondas de calor.

Desta forma, o melhor isolante será aquele que consiga quebrar a maior parte dos tipos de propagação térmica acima descritos.

Fire shelter

Equipamento usado em incêndios florestais como último recurso de sobrevivência pelos bombeiros. Frequentemente chamado de "abrigo" no nosso país, é uma espécie de uma capa feita com material resistente a agressões e à temperatura, sob a qual os bombeiros se deitam e ficam isolados das chamas caso sejam encurralados pelo incêndio.

O material que o constitui é folha de aluminio e fibra de vidro. A folha de alumínio garante-lhe uma reflexão da radiação térmica na ordem dos 95% bem como isolamento às correntes de convecção devido ao seu formato. Quanto à fibra de vidro dá uma maior resistência à folha de alumínio.

Assim, a nossa escolha acerca do isolamente recai sobre o *fire shelter* pois reduz eficazmente a propagação de calor por radiação. O caso da transmissão de calor por condução não se aplica dado que neste método o calor só é propagado se os objetos estiverem em contacto com as fontes de calor. E quanto à convecção seria sempre um problema independentemente de qual fosse o isolante, pois a forma de isolar a mesma dependeria do formato da caixa, o que não é possível alterar.

3.4 Sensores

Pretende-se captar informação do ambiente de acordo com estudos anteriores sobre incêndios florestais, os quais indicam os gases mais nocivos para a saúde humana na realidade das florestas portuguesas.

Os sensores podem ser de dois tipos, sensores digitais ou com circuito integrado ou sensores analógicos. As principais diferenças entre eles são:

- Os sensores analógicos, apenas nos permitem ler valores analógicos, tipicamente valores entre 0 e 1023, convertidos a partir da sua voltagem de saída;

- Os sensores digitais ou com circuito integrado, são sensores que vêm acompanhados por um circuitos e permitem ler diretamente um valor convertido para uma unidade de medida adquado à aquilo que se está a medir, por exemplo, num sensor de CO deste tipo podemos ler diretamente os valores convertidos em ppms. Normalmente são mais caros pois.

Sendo apenas um protótipo, e uma vez que o orçamento do projeto tem limitações, não serão adquiridos sensores demasiado caros. Desta forma iremos optar por sensores analógicos simples.

3.5 GPS

Tratando-se de um sistema que necessita de detetar e armazenar as localizações dos bombeiros ao longo do tempo em que estes se encontram no teatro das operações, é necessário recorrer ao sistema de GPS. Através deste sistema podemos saber as coordenadas (latitude, longitude e altitude) de um objeto, bem como a direção e velocidade caso este se encontre em movimento.

Para cumprimento dos objetivos do sistema integrámos nos dispositivos móveis de captura de dados, componentes GPS para detetar a sua localização.

3.6 Rede

Neste projeto pretende-se uma tecnologia de comunicação sem fios para comunicação entre vários dispositivos emissores de dados e um coordenador da rede que recebe a informação. Outros requisitos para a rede são: comunicação forte em espaços abertos (fora de edifícios) e baixo consumo energéticos. A nossa escolha é o ZigBee [25].

3.6.1 ZigBee

O ZigBee é um protocolo de comunicação bastante robusto operando sobre o padrão IEEE 802.15.4.

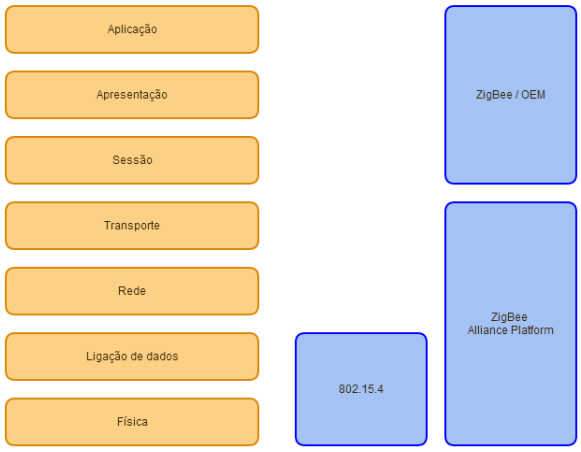
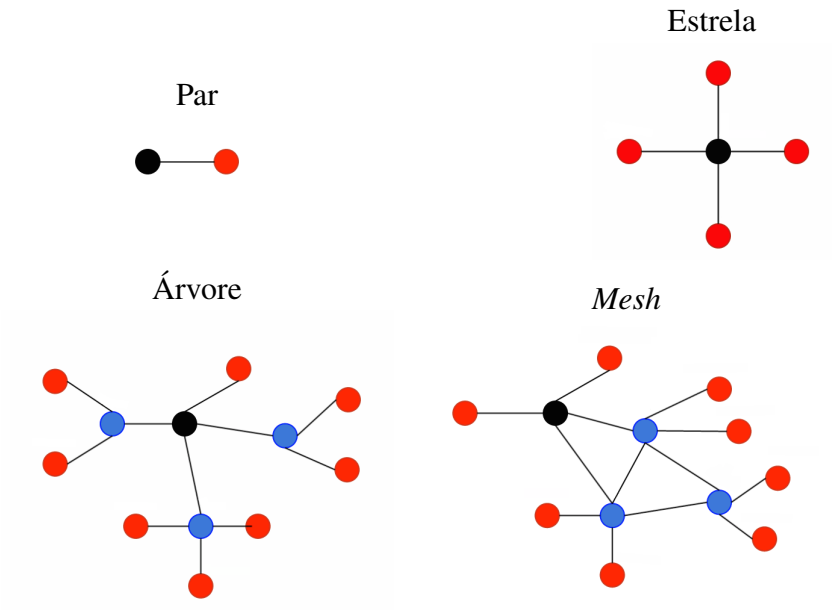


Figura 3.10: Camadas do modelo OSI e ZigBee

- **802.15.4** - É um padrão mantido pelo IEEE (*Institute for Electrical of Electronic Engeneers*) que especifica a camada física e efetua controle de acesso para redes sem fios pessoais de baixas taxas de transferência. Foca-se no baixo consumo energético, e direciona-se para redes com baixo fluxo de dados. Comparativamente

com as camadas do modelo OSI, o 802.15.4 situa-se ao nível das camadas 1 (Física) e 2 (Ligação de dados), tal como pode ser observado no esquema da Figura 3.10.

- **XBee** - São módulos de frequências de rádio que permitem comunicações no padrão ZigBee. Ou seja, definem os dispositivos físicos que permitem efetuar comunicação via ZigBee.



Legenda: Preto - Coordenador, Azul - Encaminhadores, Vermelho - Terminais

Tabela 3.2: Topologias de rede suportadas pelo ZigBee

Existem vários dispositivos que nos permitem criar uma rede ZigBee. Utilizámos o XBee Pro S2B, que possui a capacidade de comunicação dentro de edifícios até 60m e ao ar livre até 1500m, necessitando para alimentação apenas de 40mA e 3.3V.

Como o dispositivo é para ser usado em cenários onde tipicamente a distância dos bombeiros ao veículo poucas vezes atinge os 200m, podemos concluir que o dispositivo cumpre os requisitos do sistema. O Capítulo 5 apresenta resultados sobre testes feitos às distâncias de comunicação destes dispositivos.

Dentro de uma rede ZigBee podemos ter três tipo de despositivos:

- **Coordenador** - além de formar a rede, gere os endereços e outras funcionalidades, tais como a segurança da rede. Podem ainda efetuar tarefas de enviar, receber e encaminhar dados.

- **Encaminhador** - junta-se a redes existentes, permitindo enviar, receber e encaminhar dados para outros nós.

- **Terminal** - nó mais simples da rede, permitindo apenas juntar-se a uma rede existente e enviar e receber dados.

O XBee permite criar automaticamente uma rede desde que todos os dispositivos estejam pré-configurados para pertencer à mesma rede. O ZigBee pode ser configurado para dois tipos de comunicação distintos: comunicação simples e em modo API.

3.7 Aplicação cliente

Pretende-se desenvolver uma aplicação para obter informações sobre localização e que utilize os serviços *Web* disponibilizados pelo *middleware* para monitorizar os bombeiros no teatro de operações e adicionar e configurar novos serviços ao mesmo. No entanto, deve manter-se um elevado grau portabilidade, pois esta deve poder ser executada em diferentes sistemas operativos de *smartphones*, bem como em computadores.

Inicialmente, para a aplicação móvel dava-nos jeito uma aplicação nativa que pudesse ser instalada nos dispositivos. Contudo, esta traria problemas de compatilidade, sendo necessário desenvolver diferentes aplicações para os diversos sistemas operativos móveis, bem como outra para a versão de computador. Foi então que surgiu a hipótese de uma aplicação *Web*, que necessita apenas de um *browser* para ser executada, podendo facilmente ser executada em dispositivos móveis e computadores a partir do mesmo código-fonte. Em contrapartida este tipo de aplicações trazem problemas ao nível da interação com os componentes do dispositivo e funcionalidades do sistema operativo. De modo a resolver estes problemas decidiu-se optar por uma aplicação híbrida. Considera-se que estas sejam realmente mais vantajosas graças ás tecnologias usadas reduzindo bastante o tempo de desenvolvimento. Além disso, podemos ainda com o memso código-fonte exportá-la para diversos sistemas operativos além de podermos executá-la como aplicação *Web*, trazendo um pouco do melhor dos dois mundos.

Para justificar melhor a nossa escolha, são apresentados os pontos fortes e fracos de cada tipo de aplicações, segundo Stefan Von Gagern [41] nas secções 3.7.1 a 3.7.3.

Relativamoente aos requisitos funcionais da aplicação, estes são descritos pelos casos de uso da Secção 3.7.4.

3.7.1 Aplicações nativas

São aplicações que correm diretamente no dispositivo ao nível do sistema operativo, necessitando de ser instaladas no dispositivo. Este tipo de aplicações consistem em código binário que ao ser executado interage diretamente com o sistema operativo subjacente (*e.g.*, iOS, Android, Windows Phone, Blackberry), podendo aceder ao *hardware* e APIs disponíveis no sistema operativo. Tipicamente este tipo de aplicações oferece ao programador um acesso mais fácil aos sensores integrados do dispositivo.

Ainda acerca do desenvolvimento de aplicações nativas, os programadores dispõem de um ambiente de desenvolvimento específico para o sistema operativo de destino da aplicação. Contudo, estas aplicações não podem ser facilmente adaptadas para outros

sistemas operativos. A tabela seguinte sintetiza os pontos fortes e fracos deste tipo de aplicações.

Pontos fortes	Pontos fracos
Muitas opções de design	Desenvolvimento complexo
Utilização otimizada de hardware e sistema operativo	Manutenção complexa
	Dificuldade de execução em multi-plataforma

Tabela 3.3: Prós e contras de aplicações móveis nativas

3.7.2 Aplicações *web*

Atualmente todos os *tablets* e *smartphones* estão equipados com *browsers* que interpretam HTML5, CSS 3 e JavaScript. Este ambiente permite desenvolver um enorme conjunto de funcionalidades, não só através páginas *web* mas também aplicações que podem ser executadas em *browsers*.

Uma das vantagens deste tipo de aplicações são as atualizações, pois ocorre ao nível do servidor, sem que os utilizadores tenham de instalar atualizações da aplicação. Estas aplicações são mais facilmente portáveis entre diferentes sistemas operativos, uma vez que o acesso é via *browser*. Não há necessidade de desenvolver uma aplicação diferente para cada sistema operativo, como acontece com as aplicações nativas, permitindo até ser utilizadas em computadores. Basicamente funcionam como se estivéssemos a aceder a uma página *web*. A tabela seguinte sintetiza os pontos fortes e fracos deste tipo de aplicações.

Pontos fortes	Pontos fracos
Caraterísticas existentes nas linguagens *web* podem ser usadas	Executado sempre num *browser*
Baixo custo de desenvolvimento	Frequentemente menos conveniente que aplicações nativas
Atualizações fáceis rápidas e fluentes	Funcionalidades e operações offline limitadas
Ampla gama de funções disponíveis graças ao HTML5	

Tabela 3.4: Prós e contras de aplicações *web*

3.7.3 Aplicações híbridas

Estas aplicações são em grande parte baseadas em tecnologias *Web*, estando disponível também ao programador um ambiente de desenvimento em liguagem nativa. Uma parte

da aplicação é sempre em código nativo contendo as operações de arranque da aplicação, as quais são executadas diretamente no sistema operativo, funcionando também como um renderizador de HTML que permite executar as plartes da aplicação desenvolvidas com tecnologias *Web*.

Este renderizador de HTML funciona como uma ponte entre o sistema operativo e uma aplicação *Web*. Existem já renderizadores pré-definidos (*e.g.* Apache Cordova).

Este tipo de aplicações traz as vantagens de reaproveitarmos o código para executar a mesma aplicação em diferentes sistemas operativos, bem como de podermos executar a parte em HTML num *browser* podendo assim usar a aplicação em qualquer tipo de dispositvo incluindo computadores. No entanto, para utilizar as mesmas como uma aplicação para *tablet* ou *smartphone*, basta exportá-la através do renderizador, gerando diferentes ficheiros de instalação de acordo com o sistema operativo de destino pretendido, permitindo assim instalá-la no dispositivo alvo.

Os componentes HTML podem estar alojados localmente no dispositivo ou num servidor à semelhança das aplicações *Web*. Em termos de atualizações é preferível mantermos os componentes num servidor, em oposição ao mantermos os ficheiros no servidor diminui a capacidade de operações *offline*. A tabela seguinte sintetiza os pontos fortes e fracos deste tipo de aplicações.

Pontos fortes	**Pontos fracos**
Vantagens combinadas de aplicações *web* e nativas	Desempenho prejudicado (ao aceder a conteúdos *web online*)
Requere pontes entre aplicações nativas e componentes *web*	Não há atualizações frequentes (em caso de alojar localmente os componentes)

Tabela 3.5: Prós e contras de aplicações híbridas

3.7.4 Casos de uso

A aplicação deve comtemplar funcionalidades para os seguintes casos de uso:

- **Controle de exposição a gases tóxicos e altas temperaturas**

  Por forma a evitar exposições muito prolongadas a altas temperaturas e inalação de grandes quantidades de gases tóxicos, que podem provocar danos para saúde humana, é necessário controlar os dados do ambiente onde os bombeiros se encontram. Admitindo que um chefe de equipa possui um dispositivo móvel (*e.g.*, *smartphone*) com esta aplicação, este pode obter os dados mais recentes sobre a equipa à sua responsabilidade sempre que pretender.

  A aplicação cliente, é atualizada sempre que forem recebidos novos dados oriundos do *middleware*. Desta forma o chefe da equipa pode controlar as concentrações de

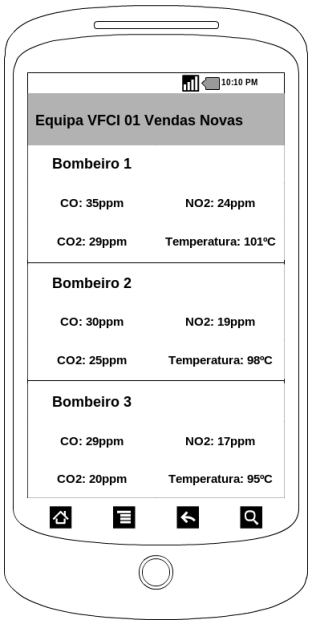


Figura 3.11: Monitorização da concentração de gases e temperatura de uma equipa de combate a incêndios florestais

gases e temperatura ambiente a que a equipa está exposta. A Figura 3.11 representa uma possível *interface* para esta funcionalidade.

- **Alerta de alta concentração de gases tóxicos**

  Durante o combate a um incêndio em mato os bomebiros de uma equipa de combate encontram-se expostos a altos níveis de monóxido de carbono. Os dispositivos enviam os dados correspondentes às concentrações dos gases.

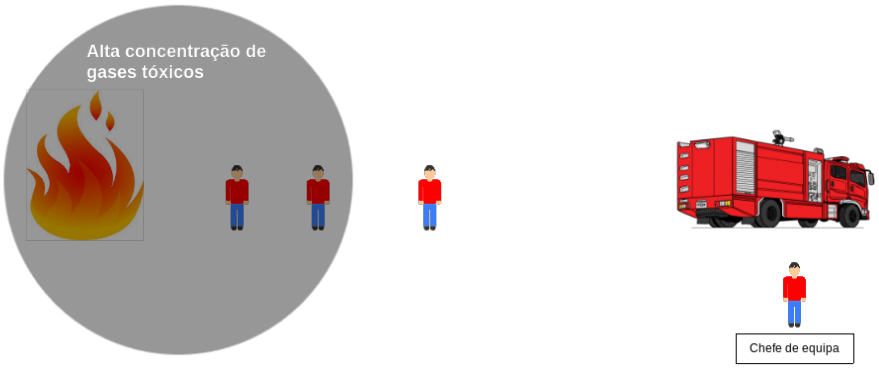


Figura 3.12: Equipa de combate exposta a altas concentrações de gases tóxicos

A aplicação ao receber os dados enviados pelo *middleware*, e verificando que estes se encontram acima dos máximos estabelecidos para as concentrações deste tipo de gás, avisa o chefe de equipa, indicando que a equipa se encontra em perigo. Este aviso pode ser realizado com a apresentação de uma caixa de diálogo semelhante à

da figura 3.13, pondendo ainda recorrer a sinais sonoros e à vibração do dispositivo de forma a alertar o seu operador.

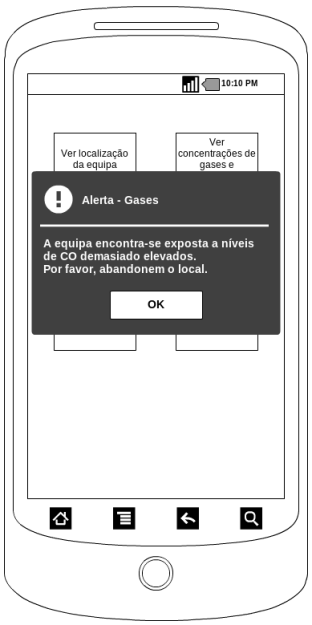


Figura 3.13: Aviso de equipa exposta a gases tóxicos na aplicação

Após o alerta é possivel abandonar o local antes que os bombeiros permaneçam demasiado tempo expostos a essa alta concentração de gases, e se desloquem para uma zona de segurança.

- **Localização de equipas**

Figura 3.14: Localização de bombeiros após fuga forçada

Nos dias de hoje já existe alguma tecnologia que ajuda a localizar os bombeiros em caso de acidente durante as operações de combate, tal como o rádio da rede do Sistema Integrado de Redes de Emergência e Segurança de Portugal (SIRESP). Só que, normalmente, só o chefe de cada uma das equipas é que possui um destes rádios, o que se torna pouco eficaz se alguns elementos se perderem durante uma fuga de emergência.

Supondo um cenário de catástrofe, em que alguns dos elementos caem inconscientes com a inalação de fumo e cansaço após uma fuga forçada e não se conseguiram juntar ao resto da equipa. É necessário socorrê-los o mais rápido possível! Mas, para isso, é necessário encontrá-los, o que actualmente se torna uma tarefa difícil. Contudo, com a ajuda do localizador GPS para cada elemento da equipa é possível saber onde cada elemento se encontra e há aproximadamente quanto tempo está imóvel no mesmo local.

Capítulo 4

Desenho e Implementação

4.1 Rede de sensores

Como já foi apresentado anteriormente, uma parte do sistema final consiste numa rede de sensores. Rede essa que compreende dispositivos de captura de dados do ambiente, os quais contêm os sensores, e dispositivos de armazenamento dos dados capturados.

4.1.1 Dispositivo de captura de dados

Este dispositivo tem como principal objetivo captar dados relativos à concentração de gases e temperatura ambiente a que os bombeiros estão expostos durante o combate a incêndios florestais bem como a sua localização GPS. Foi primeiramente implementado numa placa de prototipagem Arduino apresentada na secção 3.3.1. O desenvolvimento foi efetuado em várias fases com tarefas bem definidas previamente, sendo elas:

- Preparação dos sensores;
- Praparação dos componentes de GPS;
- Preparação dos componentes de comunicação;
- Integração de todos os componentes

4.1.1.1 Preparação dos sensores

Nesta fase foram ligados e programados individualmente cada um dos sensores. Foi a fase mais complicada e uma das mais demoradas de todo o projeto, principalmente devido à dificuldade de perceber o funcionamento de alguns deles.

4.1.1.1.1 LM35

O LM35 é um sensor de temperatura de baixo custo, e que permite ler temperaturas entre os 2 e os 150^{o}C.

Este é um sensor de muito pequenas dimensões, fácil de usar e de baixo consumo. Conta com pinos, um para alimentação que pode ir de 4 a 20V, uma para ligação à terra, e outro para leitura da temperatura. O esquema dos pinos do sensor é apresentado na Figura 4.1.

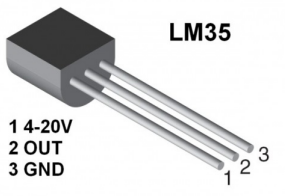


Figura 4.1: Esquema de pinos do sensor LM35
Fonte: http://craftronixlab.files.wordpress.com/2014/08/lm35_500x500_1_.jpg

A montagem do mesmo no Arduino foi feita consoante a Figura 4.2. A tempertura é calculada a partir da leitura do sensor na porta analógica do Arduino, à qual está conectado o pino 2 do sensor. Ou seja, uma vez conectado o mesmo à porta A1 do Arduino, efetua-se a leitura dos valores da porta. Por fim a temperatura é dada pelo produto do valor lido a partir de A1 com o valor 0.48828125.

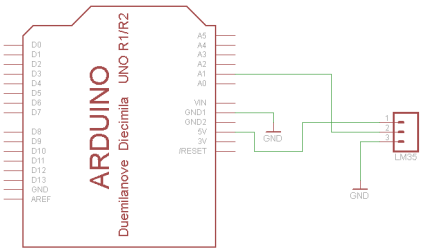


Figura 4.2: Esquema de ligação do sensor LM35

4.1.1.1.2 MQ-7

Para medir as concentrações de monóxido de carbono o sensor escolhido foi o MQ-7, por ser um sensor bastante sensível ao gás, de baixo consumo energético e com pequenas dimensões. Este sensor apresenta ainda um baixo custo.

Este foi o sensor em que houve mais dificuldades na aprendizagem do funcionamento. Apesar de ter baixa condutividade em ar limpo, o MQ-7 possui um método de funcionamento com ciclos de alta e baixa temperatura. Quando a temperatura é baixa (1.5V de tensão na resistência de aquecimento do sensor), durante 90 segundos, é feita a leitura da concentração de monóxido de carbono. Durante o ciclo de alta temperatura (5V de tensão na resistência de aquecimento do sensor), durante 60 segundos, este efetua a limpeza de outros gases absorvidos durante o período de leitura.

Uma vez ligado, o sensor necessita permanecer assim obrigatóriamente durante 48 horas, caso contrário pode gerar medições completamente erradas. Este ponto foi um dos mais críticos, pois na primeira ligação do sensor, o circuito de ligação do mesmo ao Arduino e a sua programação não foram os corretos. Segundo a tese de Diogo Cunha, em "Sistema de detecção e envio de alarmes" [23], os valores de saída esperados, durante

as primeiras 48 horas seriam semelhantes aos da Figura 4.3, contudo os valores obtidos foram os apresentados na Figura 4.4.

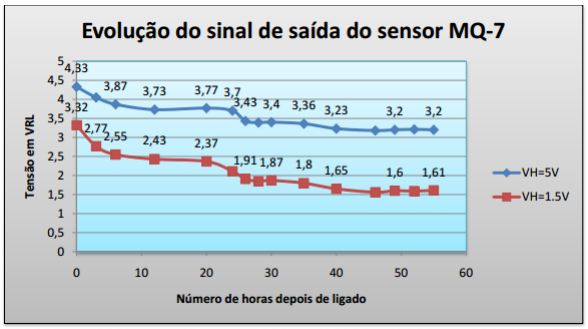


Figura 4.3: Valores esperados do sensor MQ-7

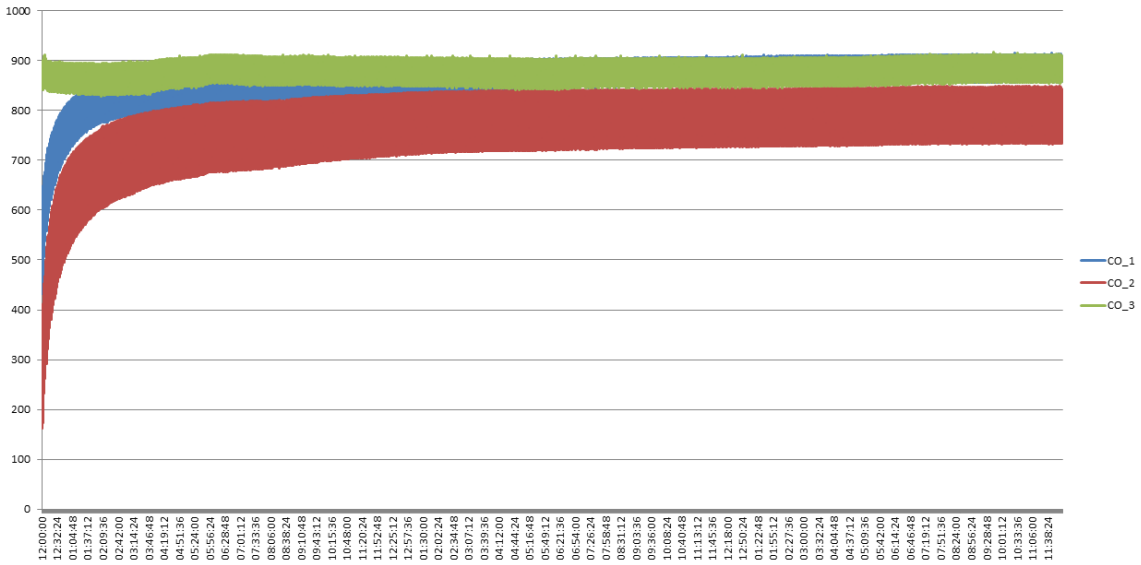


Figura 4.4: Valores lidos do sensor MQ-7

Nota: no gráfico 4.4 as séries ``CO_1'', ``CO_2'' e ``CO_3'' correspondem a três sensores MQ-7.

Apesar dos valores do eixo das ordenadas na Figura 4.3 repesentarem os valores lidos da tensão de saída do sensor e na Figura 4.4 mostrarem os valores lidos diretamente nas portas analógicas do Arduino, podemos ver que as curvas geradas por ambos os gráficos têm concavidades opostas. Contudo, pode observar-se que ambos os gráficos ao fim de algumas horas convergem para um valor fixo.

Primeiramente, foi criado um circuito simples semelhante à Figura 4.5. No entanto, foram detetadas algumas falhas no mesmo: (1) falta de condensadores para compensar os momentos de transição entre 1.5 e 5V no aquecimento do sensor; (2) intensidade de corrente demasiado elevada devido às resistências usadas para criar o circuito divisor de tensão para gerar 1.5V durante o periodo de leitura.

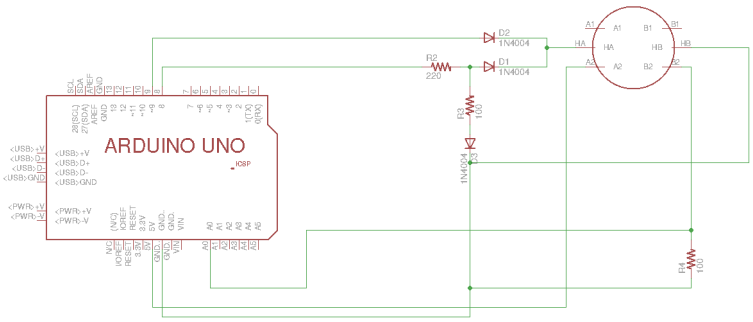


Figura 4.5: Circuito inicial para o sensor MQ-7

Assim, foi desenhado outro circuito apresentado na Figura 4.6 de modo a colmatar as falhas do primeiro. A programação dos pinos do Arduino relativamente ao circuito final é apresentada na Tabela 4.1

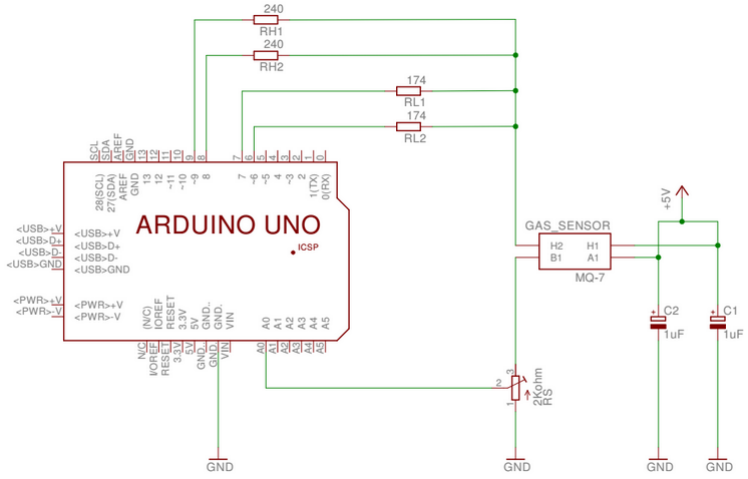


Figura 4.6: Circuito final para o sensor MQ-7

4.1.1.1.3 MiCS-2710

Para efetuar as medições de dióxido de azoto, o sensor escolhido foi o MiCS-2710. Este é um sensor de alta sensibilidade, resposta rápida, baixo consumo energético, pequenas dimensões e altamente resistente ao choque.

À semelhança do MQ-7, este sensor também opera com duas resistências internas, uma para aquecimento do sensor e outra para medição. No exterior conta com 4 pinos, o quais são apresentados na Figura 4.7.

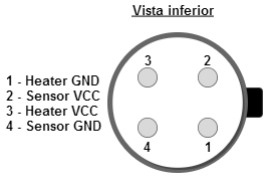


Figura 4.7: Esquema de pinos do sensor MiCS-2710

Estado do sensor	Arduino - modo das portas		Função do sensor
	Portas 8 e 9	Portas 6 e 7	
Voltagem de aquecimento (alta)	*Output*	*Output*	Aquecimento / Limpeza de gases
Voltagem de aquecimento (baixa)	*Input*	*Output*	Medição
Sem aquecimento (desativado)	*Input*	*Input*	Poupança de energia

Tabela 4.1: Configuração dos pinos do Arduino para o sensor MQ-7

Este sensor não pode ser ligado diretamente, devido a algumas restrições de corrente e tensão de funcionamento, pelo que necessita de algumas resistências de acordo com o esquema apresentado na Figura 4.8.

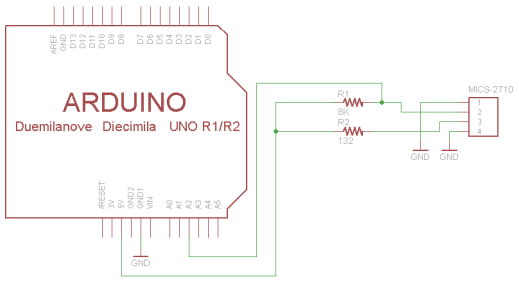


Figura 4.8: Esquema de ligação do sensor MiCS-2710

4.1.1.2 GPS

Os módulos escolhidos para detectar a localização geográfica podem ser diretamente ligados a um Arduino, funcionando com a alimentação de 5V disponibilizada pelo mesmo e trocando dados com este através de comunicação *serial* (UART). Esta comunicação pode ser feita diretamente através das portas RX e TX do Arduino. Estas, apesar de poderem ser utilizadas, podem causar problemas quando quisermos actualizar o *software* do Arduino via USB, se estiverem a ser usadas por outro dispositivo. Por isso foi abordada uma estratégia diferente de modo a libertar estas duas portas.

Segundo o sítio do Arduino, é possível utilizarmos outras portas do mesmo para comunicação *serial*, desde que estas sejam portas PWM e utilizemos a biblioteca SoftwareSerial disponível no mesmo site para *download*. Este método é bastante simples de usar, bastando escolher duas portas PWM, e no programa que necessita dessa mesma comunicação criar um objeto do tipo SoftwareSerial, indicando quais as portas que farão a função do RX e do TX. De resto, em termos de programação, tudo funciona da mesma forma relativamente à comunicação *serial* através das portas dedicadas ao efeito, possuindo o mesmo conjunto de funções.

Ainda em relação aos módulos de GPS, estes enviam constantemente para o Arduino

strings com todas as informações recolhidas. As informações enviadas pelos módulos contêm o número número de satélites ativos, coordenadas (latitude e longitude), altitude, direção, velocidade, entre outros dados. Desta forma, basta então esperar que se atinja um determinado número de satélites (mínimo 3) para obter dados minimamente fiáveis, e podermos utilizá-los.

Optou-se por incluir uma antena interna, de modo a manter fora da caixa o menor número possivel de componentes.

Na Figura 4.9 podemos observar a configuração das portas do módulo de GPS utilizado. O esquema de ligação das portas do módulos GPS às portas do Arduino é especificado na Tabela 4.2

Porta Arduino	**Porta GPS**
5V	5V (qualquer uma das duas disponíveis)
GND	GND (qualquer uma das duas disponíveis)
RX *	TX
TX *	RX

Tabela 4.2: Esquema de ligação do módulo de GPS ao Arduino
(* Podem ser substituídas por portas PWM)

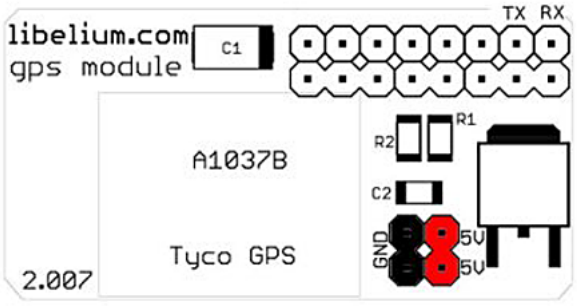


Figura 4.9: Esquema das portas do módulo de GPS [32]

A maior dificuldade encontrada nesta fase do trabalho foi o tempo que o módulo demora até encontrar o número suficiente de satélites. A primeira vez que este é ligado deveria levar cerca de 2 a 3 minutos. No entanto, uma vez que me encontrava numa área com bastantes edifícios à volta, este tempo foi claramente superior. Detectou-se que não foi apenas na primeira vez que o dispositivo foi ligado, mas sempre que o era.

4.1.1.3 ZigBee

A nossa rede ZigBee utiliza dispositivos XBee Pro Series 2B, devido ao seu alcance que pode atingir 1500 metros ao ar livre sem obstáculos pelo meio. Foram instalados tanto nos dispositivos móveis de captura de dados para envio dos mesmos, como na estação base para coordenação da rede e recepção de dados.

Durante a configuração da rede surgiram algumas dúvidas, uma vez que os dispositivos podem ser configurados de duas maneiras diferentes (API ou AT), acabando por se optar pelo modo API. A razão pela qual escolhi o modo API foi o facto de existirem bibliotecas para utilização desta tecnologia tanto para Arduino como para Java, que apenas funcionam caso os dispositivos estejam configurados desta forma. Ao contrário do modo AT em que a troca de dados entre os dispositivos funciona com simples cadeias de caracteres enviadas de uns para os outros, no modo API esta troca de dados é feita de forma mais organizada através de pacotes minimamente estruturados. Para criar uma rede é necessário ainda que todos os dispositivos estejam configurados com o mesmo identificador de rede de área pessoal (PANID), bem como com o mesmo canal de comunicação. Para poder efetuar todas estas configurações dos dispositivos foi utilizado o programa X-CTU que permite instalar o *firmware* nestes dispositivos e aplicar todas as configurações necessárias.

Quanto à distribuição das placas XBee, após a sua configuração instalou-se uma placa na estação base configurada como coordenador, por forma a criar a rede quando outros dispositivos do tipo *router* ou *endpoint* com o mesmo PANID forem ligados. Os restantes dispositivos instalados nos dispositivos de captura de dados foram configurados como *endpoint*.

Quanto à forma de comunicação dos dispositivos com o Arduino, pode ser feita da mesma forma que os módulos de GPS, ou ligados diretamente às portas RX e TX do Arduino ou a outras portas PWM que estejam disponíveis, desde que na programação do dispositivo utilizemos a biblioterca SoftwareSerial. Assim, para evitar problemas futuros caso se queira atualizar o software do Arduino optou-se por usar as portas PWM em conjunto com a biblioteca SoftwareSerial.

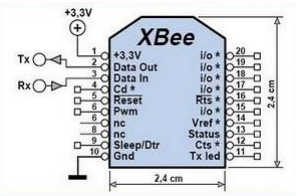


Figura 4.10: Esquema de pinos XBee Pro S2B

A configuração dos pinos destas placas é apresentada na Figura 4.10. De acordo com esta informação, a ligação das portas do Arduino aos pinos da placa é feita segundo a Tabela 4.3.

Porta Arduino	Pino XBee
3.3V	3.3V
GND	GND
RX *	TX
TX *	RX

Tabela 4.3: Esquema de ligação do módulo de GPS ao Arduino
(* Podem ser substituídas por portas PWM)

4.1.1.4 Integração de componentes

Esta fase de integração consistiu na junção de todos os componentes e sensores acima descritos num único Arduino. Os principais objetivos desta fase são: observar o comportamento de todos os componentes ligados em conjunto; detetar erros tanto a nível de circuitos como da programação; criar o programa final do dispositivo.

Numa primeira fase foram integrados apenas os sensores e analisados os valores dados por cada um deles ao longo do tempo, que podem ser observados no gráfico da Figura 4.11. Para a recolha destes dados foi usada uma aplicação desenvolvida em Java que se limita a receber, fazer *parse* e guardar os dados enviados pelo Arduino via USB.

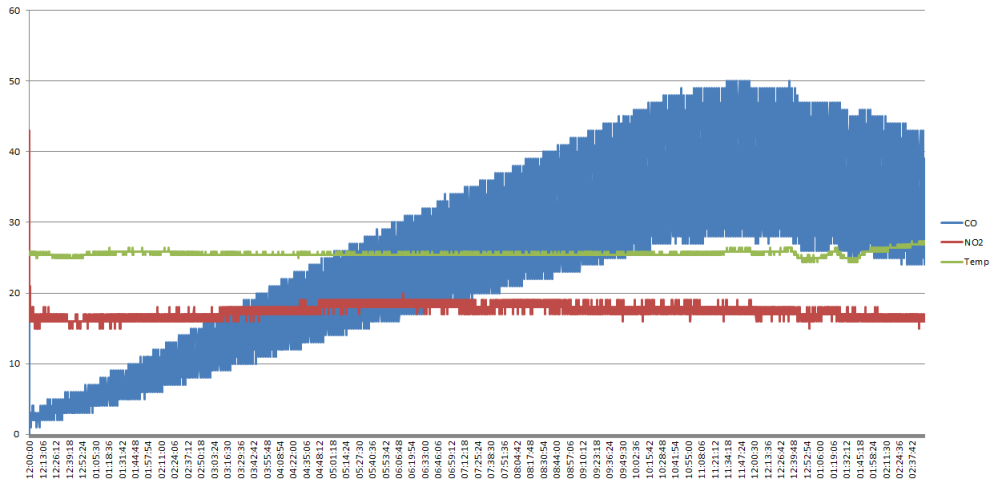


Figura 4.11: Dados recolhidos após a integração de todos os sensores

Posteriormente foram agregados ao mesmo Arduino, o módulo de GPS e a placa de rede ZigBee. Foi nesta fase que foi definido o padrão das mensagens a serem enviadas dos dispositivos móveis para a estação base contendo os dados recolhidos, padrão este apresentado no Apêndice A. Verificou-se o funcionamento da rede ZigBee através de uma outra placa de rede ligada a um computador e uma aplicação que mostrava no ecrã os dados recebidos na placa de rede enviados pelo Arduino. Na Figura 4.12 é apresentada uma fotografia do Arduino com todos os sensores e restantes componentes ligados.

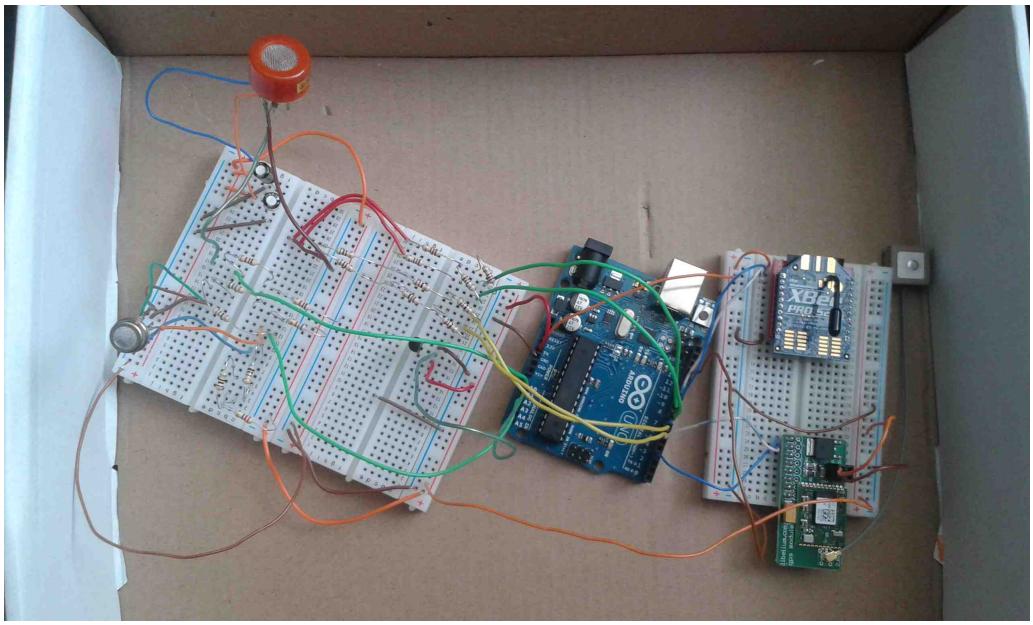


Figura 4.12: Integração de todos os componentes num único Arduino

4.1.1.5 Migração

Como pudemos ver na Figura 4.12 a utilização de um Arduino para o dispositivo móvel implicaria que este tivesse dimensões muito grandes. Isto acontece porque o Arduino contém vários componentes que não são necessários ao dispositivo final, e além disso seria necessário acoplar uma outra placa com os restantes componentes necessários ao funcionamento do dispositivo.

Assim, optou-se por criar uma nova placa de circuito impresso (PCB) contendo apenas os componentes indispensáveis ao funcionamento do dispositivo. Esta placa final irá ainda manter algumas caraterísticas do Arduino.

4.1.1.5.1 Microcontrolador

O dispositivo necessita de um núcleo que possa ser programado e tenha a capacidade de gerir e controlar os outros componentes. Desta forma, recorreu-se ao microcontrolador Atmega328P, principalmente por ser o microcontrolador utilizado pelo Arduino. Assim, é possível reutilizar o programa já desenvolvido anteriormente para o Arduino.

Em termos de circuitos foi utilizado um esquema (Figura 4.13) que esboça a correspondência entre as portas do Arduino e os pinos do microcontrolador, por forma a utilizar as mesmas portas que eram utilizadas no Arduino, evitando assim alterar o código. De resto, para funcionar, o microntrolador necessita apenas de dois condensadores para proteger do picos de corrente e uma resistência no pino "Reset".

Quanto à sua programação, é necessário antes de carregar qualquer programa, instalar o *bootloader*, e só depois é que se coloca o programa no microcontrolador. Sob pena

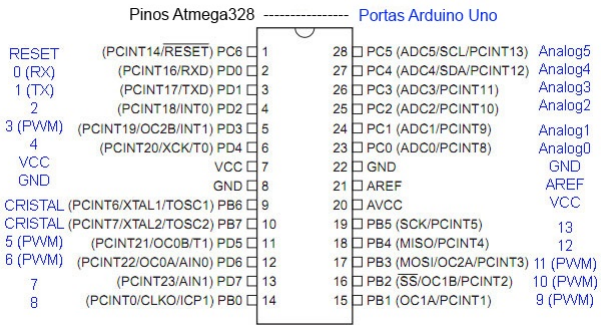


Figura 4.13: Mapeamento entre pinos do Atmega328 e portas do Arduino Uno

de existir algum problema na placa final, decidiu-se instalar o *bootloader* e carregar o programa no microcontrolador antes de ser colocado na PCB. Para este processo recorreu-se a outro Arduino, servindo de ISP para programar o microcontrolador. As instruções e circuitos necessários para efetuar estas tarefas são apresentados no Apêndice B.2, os quais foram criados com base num tutorial [3] existente no sítio do Arduino para o efeito.

4.1.1.5.2 Reguladores de tensão

Alimentando o circuito com uma pilha de 9.5V, torna-se necessário baixar a tensão para 5V, de forma a alimentar o microcontrolador, o módulo de GPS e os sensores. É também necessário reduzir a mesma para 3.3V para alimentação da placa de rede ZigBee. Para conseguir estes valores existem várias formas de o fazer tais como: criando circuitos divisores de tensão com resistências, aplicando conversores DC/DC (lineares) ou reguladores de tensão.

Uma vez que a maior parte dos componentes necessitam de 5V, e há apenas um componente a consumir menos que isso, inicialmente focou-se apenas em reduzir os 9.5V para 5V.

Os circuitos divisores de tensão podem ser aplicados segundo o esquema da Figura 4.14. Sendo que para calcular o valor das resistências utiliza-se a fórmula $V_{out} = \frac{R2}{R1+R2} * V_{in}$, tendo em conta a V_{in} que estamos a utilizar (neste caso 9.5V) e a V_{out} que pretendemos (5V). No entanto, este não é considerado o método mais eficaz para uma carga que varie pois não tem regulação, ou seja, se o circuito gastar mais a tensão de alimentação vai baixar, se o circuito gastar menos do que o planeado a tensão de alimentação vai subir. Ora, tendo em conta que a carga não vai ser sempre a mesma, visto que o sensor MQ-7 tem períodos de maior consumo que outros esta opção foi afastada.

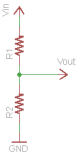


Figura 4.14: Circuito divisor de tensão

Quanto aos conversores DC/DC estes podem ser de vários tipos (*Step-up*, *Step-down* ou mistos) e devem ser utilizados quando: (1) o circuito a alimentar tem um consumo elevado; (2) a tensão de entrada é muito diferente da tensão de saída; (3) o circuito é alimentado por baterias, principalmente se a tensão da bateria for menor que a tensão de saída (através de conversores *Step-down* ou mistos).

Em relação aos reguladores de tensão, são baratos e muito simples de usar, consistindo num componente, e podem ser usados qunado se necessita de uma voltagem constante de saída. São circuitos de baixo consumo e a tensão de entrada é superior à de saída mas não em demasia.

Assim, optou-se pelos reguladores de tensão, uma vez que são aqueles que mais se adaptam ao nosso tipo de circuito. Para conseguir ter as duas tensões de saída pretendidas (5V e 3.3V) optou-se por colocar os reguladores em cascata, ou seja, um primeiro regulador passa a tensão de 9.5V para 5V, e um segundo reguladar passa de 5V para 3.3V.

Os reguladores escolhidos foram LM1117 para baixar de 9.5V para 5V, uma vez que é o mesmo regulador usado pelo Arduino Uno, e o MCP1702 sendo que o LP2985-33 (o utilizado pelo Arduino Uno) só era possível obter o modelo para soldar na face, o que não é desejável admitindo que os componentes são soldados manualmente.

4.1.1.5.3 Placa de circuito impresso (PCB)

Após a escolha e estudo dos componentes necessários para o dispositivo, foi elaborado um esquema do circuito englobando todos esses componentes e sensores pretendidos. Com base neste esquema foi posteriormente desenhada uma placa de circuito impresso (PCB) com o circuito do dispositivo, a qual é a base do dispositivo onde são ligados os sensores. Tanto o esquema do circuito como o desenho da placa podem ser observados no Apêndice B.1.

A placa foi desenhada de forma a que o dispositivo possar ser reprogramado através de pinos que podem ser ligados a um Arduino que sirva de ISP ao mesmo. É possível ainda que tanto os sensores como os módulos de GPS e as placas de rede ZigBee possam ser substituidos pois não estão soldados diretamente na placa mas sim conectados através de *sockets* que permitem a sua fácil substituição.

4.1.1.5.4 Caixa de proteção e isolamento

Tanto a PCB do dispositivo como a placa de rede, o módulo de GPS e a pilha que alimenta o dispositivo foram fixados dentro de uma caixa de plástico, a qual foi isolanda com *Fire Shelter*, por forma a melhorar o isolamento térmico e proteger os componentes existentes no seu interior, tal como já tinha sido referido na Secção 3.3.2.

No entanto, no que diz respeito aos sensores, estes não podem ficar completamente

fechados dentro da caixa sob risco de não conseguirem efetuar medições fiáveis. Portanto, foram abertos orifícios num dos topos da caixa (frente) à medida de cada um dos sensores de maneira a que estes mantenham a superficie de sensoriamento não isolada e o restante volume do sensor dentro da caixa, tal como pode ser observado na fotografia da Figura 4.15.

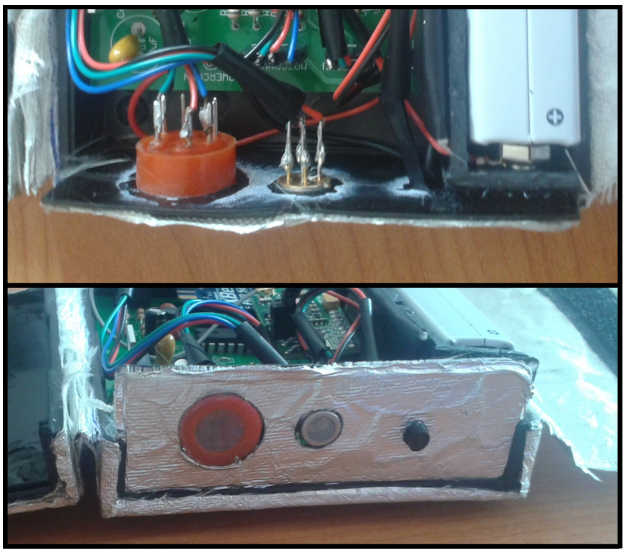


Figura 4.15: Caixa com sensores

Foi ainda adicionado no topo da caixa oposto aos sensores (trás) um botão que permite ligar e desligar o dispositivo, com o objetivo de poupar bateria.

Numa das laterais foi colocado um fecho de velcro com o objetivo de permitir a abertura da caixa para troca de pilhas ou algum componente ou sensor danificado. Do lado oposto foram implatadas duas percintas em velcro também, para capacitar o transporte do dispositivo no cinturão do bombeiro.

4.1.2 Estação base

De forma a criar um dispositivo que consiga receber e armazenar dados dos dispositivos que efetuam as medições dos dados do ambiente, assim como com a capacidade de fornecer serviços a aplicações cliente baseado numa arquitetura orientada a serviços (SOA), foi necessário desenvolver uma camada de *software* robusto e flexível a diversos tipos de tecnologias. E tal como abordado anteriormente este *software* foi adaptado de um outro já existente, o MuFFIN.

Além desse *software* foi necessário ainda configurar um conjunto de serviços do sistema operativo, para poder estabelecer comunicação via *WiFi* com as aplicações, bem como inicializar o nosso *middleware* logo após o arranque do sistema.

4.1.2.1 *Middleware*

Para melhorar o desempenho do *middleware* existente em dispositivos de computação miniaturizados, substituiu-se a camada de acesso aos dados bem como a infraestrutura de comunicação entre os componentes do *software*.

4.1.2.1.1 Camada de acesso aos dados

Nesta fase foi refeito o componente de *software* que faz a gestão dos dados entre a base de dados e o *middleware*.

Dentro desta camada foram criados alguns módulos e submódulos de forma a organizar e melhorar a capacidade de extensibilidade do *software*. Os principais módulos são:

- **cms.dbAccess**: Contém a *interface* da camada de acesso aos dados para que estes possam ser acedidos pelas camadas de níveis superiores. Além disso dispõe de um componente muito importante, o "RepositoryFactory", que gere os repositórios em memória de cada uma das tabelas da base de dados. Este componente rege-se pelo padrão *Factory Method*, fazendo com que esses repositórios inicialmente não existam e apenas sejam criados à medida que são necessários.

- **cms.dbAccess.Exception**: Agrega todas as excepções relativas aos erros que possam ocorrer durante a execução de funções desta camada.

- **cms.dbAccess.adts**: Responsável pelos tipos de dados abstratos e mais genéricos desta camada. Aqui foram criados:

  - "AbstractMapper": *Mapper* abstrato que aplica o padrão *Data Mapper* de forma genérica através da combinação com o padrão *Template Method*. Ou seja, estabelece definitivamente as operações que devem ser iguais em todos os *mappers* para a base de dados, além de impor um conjunto de métodos que devem ser definidos posteriormente quando for efetuada a implementação de cada um dos *mappers*.
  - "RDBMap": Implementa a *interafce* da estrutura de dados Map, de forma a que interaja diretamente com a base de dados. Isto é, cada vez que é efetuada uma operação de inserção, leitura ou remoção da estrutura Map, estes dados são simultaneamente escritos, lidos ou eliminados da base de dados respetivamente. Este componente contém um estrutura de dados do tipo WeakHashMap que mantém em memória os últimos dados a serem acedidos, com o objetivo de poupar tempo de leitura do disco quando acedemos muitas vezes a um determinado objeto. WeakHashMap contempla ainda a caraterística de utilizar referências fracas, o que quer dizer que assim que deixam de haver refências para um determinado objeto, o mesmo é apagado da memória.

  - "DataSource": Estabele as configurações necessárias para se poder aceder a um determinado repositório de dados. Ou seja, mais uma vez se recorreu ao padrão *Template Method*, de maneira a que, quando quisermos mudar de tipo de base de dados, basta extender esta classe e definir apenas os métodos necessários para aceder à mesma.
  - "Proxy": Estabelece um tipo de dados genérico para as *proxies* a serem usadas por cada tipo de dados em concreto (*e.g. Sensor, Observation*), uma vez que vamos recorrer ao padrão *Lazy Load* e é necessário termos definida uma *proxy* para cada tipo de dados.
  - "StatementsFactory": Indica todas as *queries* SQL necessárias para o bom funcionamento desta camada. Uma vez adicionada uma nova fonte de dados (uma nova implementação de "DataSource") deve ser criada também uma implementação de "StatementsFactory" correspondente.
  - "SimpleRDBList" e "ComplexRDBList": Semelhante ao "RDBMap", mas aplicado à estrutura de dados do tipo List, na medida em que todos as inserções e remoções da estrutura de dados são simultâneamente efetuadas na base de dados. Estas duas estruturas de dados aplicam então o padrão *Forign Key Mapping*, e ainda o *Association Table Mapping* no caso da segunda. O principal objetivo é gerir no paradigma orientado a objetos as associações de um-para-muitos no caso da "SimpleRDBList", e de muitos-para-muitos no caso da "ComplexRDBList".

- **cms.dbAccess.SQLite**: Implementa os tipos abstratos "DataSource" e "StatementsFactory" para bases de dados do tipo SQLite, uma vez que é a base de dados que utilizamos para armazenamento dos dados do *middleware*.

Existem ainda dois submódulos com alguma extensão direcionados, mais propriamente, ao funcionamento do *middleware* e aos serviços disponibilizados:

- **cms.dbAccess.services**: Acopla os tipos de dados relativos aos serviços do *middleware* (*e.g.* módulos, intâncias de serviços, *gateways*) e *interfaces* definidos para acesso aos mesmos a partir do exterior. Para cada tipo de dados definido existem respectivas implementações, *proxies* e *mappers*.

- **cms.dbAccess.sos**: Reúne todos os tipos de dados necessários ao funcionamento do componente do Serviço de Observação de Sensores (SOS), contando também com uma implementação, uma *proxy* e um *mapper* para cada tipo de dados existente, bem como interfaces para acesso aos respectivos dados a partir do exterior do submódulo onde estão inseridos.

4.1.2.1.2 Integração de compoenentes de *software*

Uma vez decidido o tipo de infraestrtura para dar suporte à passagem de dados entre serviços passou-se à sua implementação. Desenvolveu-se um componente do tipo central de eventos. A arquitetura do mesmo é composta essencialmente por três conceitos: eventos, serviços e central de eventos. O diagrama de classes e módulos do mesmo é apresentado na Figura 4.16.

No que diz respeito aos eventos, estes são representados pela classe "Event", que por sua vez faz uso da classe "TextMessage", usada para estruturar as mensagens. Esta classe "Event" contém ainda um atributo "source" que indica qual o serviço que criou o evento.

Em relação aos serviços, estes devem seguir uma implementação do tipo "AbstractService", o qual contém um determinado conjunto de funcionalidades implementadas, genéricas para todos os serviços que venham a ser criados. Contudo, esta mantém ainda um conjunto de funcionalidades (métodos) que devem ser definidas aquando da implementação do serviço, de acordo com o propósito e a finalidade do mesmo. Podemos observar mais uma vez o uso do padrão *Template Method*.

Por último, a central de eventos recebe os eventos de todos os serviços e assegura que faz chegar os mesmos a todos os interessados. Ou seja, contém uma fila de eventos pendentes, os quais serão entregues aos interessados através de uma *hashmap* de subscrições, onde para cada chave (que corresponde ao serviço que criou o evento) contém uma lista de todos os serviços que subscreveram o evento correspondente à chave. Posteriormente o evento será enviado para todos os serviços existentes nessa lista, os quais irão efetuar todas as operações pretendidas para o mesmo, e por fim o evento será entregue de novo à central. A central possui ainda uma *hashmap* com os serviços que necessitam receber todas os eventos de todos os outros serviços.

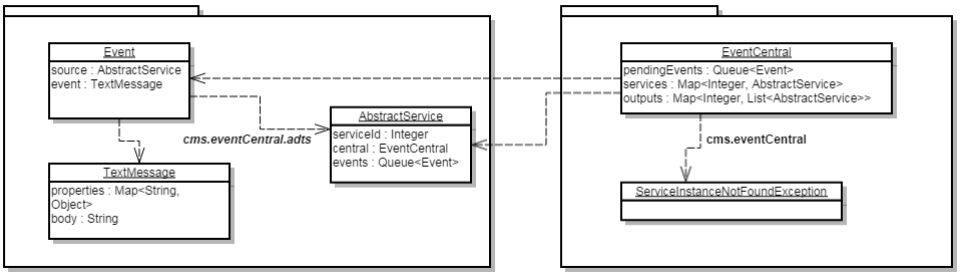


Figura 4.16: Diagrama de classes e módulos do componente EventCentral

4.1.2.1.3 Serviços disponibilizados

O propósito atual do *middleware* é bem mais específico que a versão do MuFFIN que pegámos inicialmente para efetuar as alterações necessárias. Por isso foi preciso também

adicionar alguns novos métodos aos serviços na *Web*. Esses novos métodos foram criados para dar resposta a alguns dos requisitos da aplicação cliente.

Assim, foi possível implementar um sistema parcialmente baseado na arquitetura SOA. Parcialmente porque a arquitetura SOA é aplicada na parte da comunicação entre o *middleware* e as aplciações cliente, sendo que a comunicação com os dispositivos de captura de dados foi implementada segundo uma abordagem diferente.

No Apêndice E são apresentados os serviços disponibilizados pelo *middleware* bem como a documentação correspondente.

4.1.2.2 Configurações do sistema

4.1.2.2.1 Configuração de rede *WiFi*

Para os serviços poderem ser utilizados pela aplicação cliente é necessário que o dispositivo onde estes estão instalados esteja acessível através da rede. Tratando-se de equipamentos para ser utilizados em trabalho e uma vez que o dispositivo que fornece os serviços se encontra instalado no carro, esta rede teve que ser obrigatóriamente sem fios. Sabendo ainda que nos locais onde os dispositivos serão utilizados não existem redes *WiFi* para os dispositivos se ligarem, surge a necessidade de criar uma rede deste tipo para fazer com que o sistema funcione corretamente.

Para evitar colocar um *router* dentro dos veículos de combate a incêndios, uma vez que se tornava demasiado equipamento a ser instalado, traria maiores custos e ocuparia bastante espaço, optou-se por transformar também o dispositivo instalado no veículo num *router WiFi*. Este processo foi efetuado recorrendo a um simples adaptador *WiFi* conetado via USB e configurando o sistema de forma a que este em vez de permitir a ligação do dispositivo a outras redes permita que outros dispositivos se liguem a este como se se tratasse de um *router WiFi*.

Para conseguir isto foram necessárias duas aplicações:

- **hostapd**: É uma aplicação para configuração de pontos de acesso e servidores. Permite estabelecer os dados de autenticação do ponto de acesso a ser criado. Esta aplicação implementa gestão de pontos de acesso 802.11 (*wireless*), autenticação IEEE 802.1X/WPA/WPA2/EAP, servidor EAP, entre outras funcionalidades relacionados com o *Remote Authentication Dial In User Service* (RADIUS).

- **isc-dhcp-server** : É uma aplicação de servidor *Dynamic Host Configuration Protocol* (DHCP). Possibilita a atribuição de endereços IP automaticamente quando um novo dispositivo se liga na rede criada pelo dispositivo onde esta está a ser executada.

Assim, com o hostapd fornecemos um ponto de acesso via *WiFi*, simulando a existência de um *router* que na verdade é a nossa estação base (Raspberry Pi). Após a ligação de

um utilizador a esta rede é-lhe atribuído um endereço IP que pertença a essa mesma rede através do isc-dhcp-server.

4.1.2.2.2 Arranque automático dos serviços

Ao longo de todo o desenvolvimento do trabalho, cada vez que era necessário iniciar o *middleware*, acedia-se ao dispositivo pela linha de comandos através da rede, e aí eram executados os comandos para execução do *middleware*. No entanto, durante um incêndio florestal, ou no veículo durante a viagem até ao incêndio, não dispomos de computadores para iniciar o *middleware*. Desta forma surgiu a necessidade de iniciar o *middleware* logo após o arranque do sistema operativo.

A forma mais fácil de iniciar o *middleware* é criar um serviço que efetue os comandos de inicialização do mesmo. Assim, criou-se um *shell script* com os comandos necessários para garantir a inicialização correta do mesmo, bem como com os comandos para a sua execução visto que pode vir a fazer falta até mesmo para possíveis atualizações futuras ou correção de erros.

Como seria de esperar a criação deste *script* por si só não chega. É necessário, no fim, criar um serviço do sistema ligado a esse *script* e registá-lo na lista de serviços a serem inicializados pelo sistema operativo, logo após o arranque. Deste forma, conseguiu-se transformar facilmente o nosso *middleware* num serviço do sistema que é iniciado automaticamente, evitando assim ter que haver um computador ligado e conectado à estação base através da rede para iniciá-lo.

4.2 Aplicação

Como especificado no capítulo anterior, optou-se por uma aplicação híbrida. Esta opção de desenvolvimento foi vantajosa tendo em conta a escassez de tempo e ambição de desenvolver uma aplicação que pudesse ser instalada num dispositivo móvel e ao mesmo tempo executada num computador. Outra vantagem foi ao nível da programação e do *design* da mesma, pois o facto de ser desenvolvido com linguagem *Web* (*e.g.* HTML, JavaScript e CSS) facilitou o processo de desenvolvimento e tornou-o mais rápido.

Através do uso da aplicação Apache Cordova [26], é possível converter o código da aplicação desenvolvida com linguagens *Web* para aplicações que podem ser instaladas nas mais variadas plataformas. Basta para isso instalar as bibliotecas para as plataformas pretendidas e exportar a aplicação para essa mesma plataforma. Ou seja, no caso da exportação para Android, basta ter instaladas as bibliotecas necessárias para as versões do android pretendidas e a aplicação cria um ficheiro com a extensão "apk", de forma a que este possa ser instalado no dispositivo alvo. O único entrave à exportação das aplicações para outros sistemas operativos é o facto de que para exportar uma aplicação

para Windows Phone, este processo precisa ser efetuado num computador com o sistema operativo Windows 8 ou superior, bem como para exportar para uma aplicação para iOS o processo precisa de ser efetuado num computador com o sistema operativo MacOS.

Em termos de *frameworks* utilizadas, há a salientar o JQuery Mobile [31] uma vez que é compatível com o Apache Cordova. Esta *framework* está otimizada para dispositivos do tipo *touch* como *tablets* ou *smartphones* e tem como objetivo ser compatível com uma ampla variedade de dispositivos.

Com o JQuery Mobile é possível usufruir por exemplo de: (1) *widgets* otimizados para o toque; (2) compatibilidade tanto com as maiores plataformas móveis, como com os *browsers* mais famosos; (3) semelhanças com o JQuery pois foi construído a partir dele, o que gera uma pequena curva de aprendizagem desde que se esteja familiarizado com o JQuery; (4) alguma variedade de temas já disponíveis para o *design* da aplicação; (5) navegação com Ajax; (6) transição animada entre páginas. Relativamente à aplicação desenvolvida, esta foi exportada para Android, e criada uma forma de poder ser executada num computador ocultando tanto quanto possível o facto de estar a ser executado através de um *browser*.

4.2.1 Funcionalidades

Para cumprir os objetivos propostos, a aplicação desenvolvida conta com as seguintes funcionalidades:

- **Sincronização com o servidor**: Como os dispositivos que operam como estação base do sistema, os Raspberry Pi instalados nos veículos, não se encontram permanentemente ligados, a hora do sistema não é atualizada durante esse período. Uma vez ligado, a hora do sistema é a hora do momento em que o mesmo foi desligado, surgindo então a necessidade de criar uma funcionalidade na aplicação para atualizar a hora da estação base de acordo com a hora do dispositivo onde a aplicação cliente se encontra a ser executada. No fundo esta operação de sincronização com o servidor, não é mais que recorrer a um serviço *Web* disponibilizado pelo *middleware*, que atualiza a hora do sistema operativo. Para se obterem dados corretos e evitar erros no sistema, esta operação deve ser executada cada vez que a estação base é ligada.

- **Sistema de localização**: Depende do dispositivo, caso este tenha GPS ou não. Esta funcionalidade apresenta na página inicial da aplicação a latitude, longitude e altitude a que o mesmo se encontra. Caso este esteja em movimento é possível obter ainda a direção para onde se desloca e a sua velocidade. No caso das coordenadas (latitude e longitude) é possível ainda convertê-las para vários formatos. Esta funcionalidade está desativada para a versão de computador da aplicação.

- **Iniciar/Escolher ocrrência**: Para que o sistema possa guardar os dados recolhidos na rede de sensores é necessário que estes fiquem associados a uma ocorrência aberta. No modelo de dados do sistema a ocorrência é representada pelas *offerings*. Ou seja, uma vez que todas as observações necessitam estar associados a uma *offering* para serem guardadas, e estas possuem dois atributos relativos ao tempo de início e fim, tornou-se a forma mais simples de gerirmos ocorrências aproveitando desde logo as *offerings* do SOS. Ora, uma ocorrência está aberta quando a mesma não tem definida uma data de fim, contendo apenas uma data de inicio. Desta forma, quando uma ocorrência é criada, é definido apenas o atributo relativo ao tempo de início, deixando em branco o tempo de fim da mesma. Após a criação de uma ocorrência, a aplicaçao mantém logo essa como sendo a ocorrência selecionada para monitorização de dados. Posteriormente, todos os dados recebidos da rede de sensores serão associados às ocorrências abertas.

  No entanto, pode dar-se o caso do utilizador da aplicação estar a monitorizar a equipa através da aplicação e fechá-la acidentalmente. Quando voltar a abrir, a aplicação não terá nenhuma ocorrência selecionada, o que o impossiblitará de monitorizar a equipa. De modo a resolver este problema foi criada também a opção de listar as ocorrências abertas, para que possa ser selecionada a que se pretende monitorizar. Após selecionar a mesma, é possível continuar a monitorizar a equipa. A listagem de ocorrências abertas é também feita tendo em conta as *offerings* sem data de fim definida.

  Nota: Não deve existir mais que uma ocorrência aberta em simultâneo no mesmo sistema sob perigo de criar dados repetidos, e causar lentidão no sistema por processamento de operações desnecessárias.

- **Fecho de ocorrência**: No final de cada ocorrência, esta deve ser encerrda também no sistema, de maneira a que não sejam associados mais dados a essa ocorrência. Sucintamente esta operação limita-se a definir uma data de fim para a *offering* correspondente à ocorrência selecionada.

- **Monitorização**: Sempre que exista uma ocorrência selecionada na aplicação, é possível monitorizar as observações efetuadas em cada um dos bombeiros equipados com o dispositivo de captura de dados nessa ocorrência. A monitorização é efetuada individualmente mostrando apenas os dados relativos a um bombeiro de cada vez. Consite num gráfico actualizado ao longo do tempo, que mostra as oscilações das observações relativas à temperatura ambiente e gases. Permite ainda que tocando nos pontos do gráfico sejam visualizados os valores exactos de cada observação.

- **Mapa**: Muito semelhante à opção de monitorização de temperatura e gases na me-

dida em que recorre ao mesmo serviço *Web* para aceder aos dados dos bombeiros, assim como pelo facto dos dados serem atualizados ao longo do tempo. Com esta função existente na aplicação é possível visualizar no mapa a localização atual do utilizador da aplicação, bem como a última localização registada de cada um dos bombeiros que estão a utilizar o dispositivo de captura de dados. O utilizador e os bombeiros aparecem no mapa com marcadores diferentes, e clicando em cima dos marcadores podemos ter informação a que bombeiro pertence.

- **Informações de módulos**: Permite apenas visualizar todos os serviços e *gateways* instalados no sistema, bem como aqueles que se encontram em execução. Ainda em relação aos serviços já instanciados (em execução), é possível visualizar as dependências de cada um, ou seja, os serviços que cada um necessita subscrever para que possa ser executado e desempenhar a sua função corretamente.

- **Instalar *gateway***: Adiciona a possibilidade de instalação de um novo *gateway* permitindo consequentemente agregar à rede de sensores atual outros dispositivos com tecnologias de comunicação diferentes daquelas que estão a ser usadas.

  Por exemplo: Imaginando que atualmente todos os dispositivos enviam os dados para a estação base através de ZigBee, para os quais já dispomos de um *gateway* que efetua a recepção dos dados; se quiséssemos juntar à mesma rede um conjunto de dispositivos que comunicassem através de *WiFi*, bastava instalar um novo *gateway* que efetuasse a gestão dos *sockets* e receção dos dados enviados pelos dispositivos.

  Estes *gateways* devem ser ficheiros "jar", onde a classe principal seja uma extensão do tipo "AbstractThing" existente no *package* "cms.thingsGateway.SDK". "AbstractThing" por sua vez já é uma extensão do tipo "AbstractService". Esta última herança serve para que os *gateways* possam ser incluídos na central de eventos e troquem messagens com os restantes serviços do sistema. Após a sua instalação, ao contrário dos restantes serviços, os *gateways* são instanciados automaticamente.

- **Instalar serviço**: Os serviços são classes em Java que devem obedecer a um determinado conjunto de parâmetros e que se propõem a resolver um determinado problema ou tratar os dados oriundos da rede de sensores. Os parâmetros que devem respeitar estão especificados na classe "AbstractService" a qual deve ser superclasse de todos os serviços instalados.

  A instalação destes serviços faz-se através do código fonte do mesmo, sendo a sua compilação efetuada em tempo de execução do *middleware*.

- **Instanciar serviço**: Após a instalação de um serviço, o mesmo não é instanciado, necessitando de ordem do utilizador para que este seja instanciado. É durante este processo de instanciação de um serviço que o código do mesmo é compilado caso

ainda não tenho sido executado desde que o *middleware* iniciou. Pode ainda existir mais do que uma instância do mesmo serviço.

Durante a instanciação além do identificador do serviço que se pretende executar, há ainda que criar um pequeno documento no formato XML, onde são indicadas as dependências da instanciação. Ou seja, para uma determinada instância de um determinado serviço quais são as instâncias de outros serviços que esta deve subscrever na central de eventos.

4.3 Considerações finais

Dados os objetivos para a rede de sensores e dispositivos de captura de dados do ambiente, pode-se considerar que foram todos cumpridos apesar de alguns contratempos e dificuldades encontradas. A não implementação do sensor PM 2.5 na estação base, o qual foi identificado nos objetivos como um requisito, deveu-se pelo facto do produto se encontrar esgotado quando foi feita a encomenda dos sensores. E uma vez que a encomenda dos sensores levou bastante tempo a estar disponível devido a problemas burocráticos, optou-se então por não incluir o mesmo no protótipo. No entanto, este pode ser adicionado ao sistema, visto que para adicionar um novo dispositivo à estação base basta conectá-lo e instalar no *middleware* um *gateway* que trate da programação necessária para obter os dados do sensor.

Relativamente ao sensor MQ-7 foi um dos que trouxe mais dificuldades devido ao seu modo de funcionamento muito específico. Outra dificuldade encontrada e ainda relacionada com a captura dos dados está diretamente relacionada com os porblemas encontrados na Secção 4.1.1.2, onde na fase de integração de todos os componentes no mesmo Arduino, o tempo de espera até encontrar satélites de GPS era demasiado grande. Por vezes levou-nos a pensar que existiam erros de programação, dado que a inicio nunca eram apresentadas coordenadas.

De um modo geral, pode dizer-se que, de todo o sistema, o que gerou realmente mais dificuldades foi deveras a eletrónica e tudo o que esta engloba, pois as bases de conhecimento de que dispunha para resolver certos problemas nos circuitos não eram suficientes. Após muito esforço, tenacidade e algumas ajudas, foi possível então desenvolver a eletrónica dos dispositvos de captura de dados.

Em relação à aplicação cliente, de modo a aumentar o desempenho, à medida que a aplicação era desenvolvida era feito *debug* através do *browser*, evitando assim compilar a mesma e instalá-la no *smartphone*, poupando tempo. O maior problema nesta fase surgiu então quando se começou a integração da mesma com os serviços na *Web*, em que eram sempre obtidas mensagens de erro no *browser* enquanto que, quando era testada no *smartphone*, tudo corria bem. Até que se descobriu que a maioria dos *browsers*, internamente, antes de qualquer pedido HTTP com os métodos GET ou POST, enviam

um pedido com o método OPTIONS. Este método serve para verificar, antes de fazer os pedidos GET ou POST, se o servidor está ativo e a responder. Contudo, o nosso *End-point* em Java, o qual disponibiliza os serviços na *Web* não suporta pedidos HTTP com o método OPTIONS. A forma de contornar este problema foi criar um *script* que inicializa a aplicação através do *browser*, mas desativando as definições de segurança do mesmo.

Capítulo 5

Avaliação do sistema desenvovido

Este capítulo apresenta um conjunto de testes efetuados ao sistema e a análise dos resultados dos mesmos. Para cada um deles são apresentadas as respetivas conclusões.

Testar o sistema que desenvolvemos, composto por vários módulos complexos, é uma tarefa desafiante. O objetivo principal desta fase foi garantir o máximo de qualidade do sistema, evitando que o mesmo falhe e garantir que se encontra protegido contra os riscos enunciados no início do projeto.

Para explicar melhor alguns dos testes que foram levados a cabo, são apresentadas fotografias do ambiente em que foram efetuados. Os resultados são apresentados através de tabelas e gráficos. Todos os gráficos e cálculos foram realizados na aplicação Microsoft Excel 2010.

5.1 Teste de comunicação através do isolante

Figura 5.1: Teste de comunicação da rede ZigBee

É essencial proteger os componentes eletrónicos das temperaturas a que estão sujeitos em cenários de incêndios. Contudo, existe o risco da perda de comunicação. Por isso foi feito

um teste para avaliar as capacidades de comunicação entre os dispositivos sob influência dos isolantes escolhidos.

Descrição: Foram feitos dois testes utilizando uma placa Arduino com uma placa XBee configurada como Terminal da rede ZigBee. Num dos testes, as placas encontravam-se dentro de um caixa de plástico, no outro, dentro de uma caixa de plástico isolada com o *fire shelter*. Em ambos os testes a placa Arduino encontrava-se programada para enviar uma mensagem através da rede ZigBee para o nó Coordenador com um intervalo de 1000 milisegundos.

O Coordenador ficou colocado no ponto fixo enquanto a placa Arduino esteve colocada em várias distâncias em relação ao Coordenador. Assim, as distâncias testadas entre os dois dispositivos foram todos os múltiplos de 20 metros entre 20 e 140 metros, para cada um dos testes. Em cada uma das distâncias medidas o Coordenador da rede contou o número de mensagens recebidas durante um minuto.

Para realizar este teste foram necessários dois intervenientes. Em termos de equipamento foi ainda necessário recorrer a rádios de comunicação e a um odómetro para medir as distâncias.

Resultados

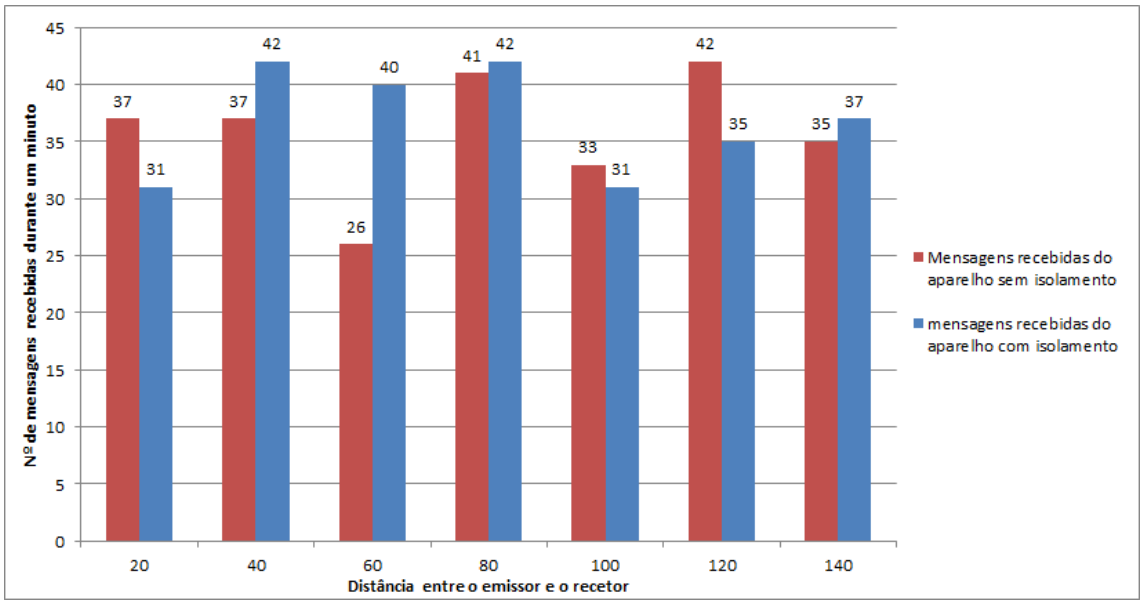


Figura 5.2: Gráfico de resultados da comunicação protegido por isolantes

Análise/Conclusões

Como podemos observar no gráfico da Figura 5.2, até aos 140 metros, o isolamento com *fire shelter* não causa intereferências significativas à comunicação comparativamente ao teste feito com a caixa de plástico. Não foram testadas maiores distâncias pois para um protótipo inicial não se julga ser necessário, uma vez que a distância dos bombeiros ao veículo, onde se encontra o recetor de dados, raramente excede os 140 metros.

Assim, conclui-se que o isolamento do dispositivo com a caixa de plástico e *fire shelter* não causa problemas à comunicação na rede ZigBee, podendo comunicar-se livremente ao ar livre até uma distância de 140 metros testada.

5.2 Teste de temperatura e isolamento térmico

Figura 5.3: Teste de temperatua dentro e fora do *fire shelter*

De forma a aferir a proteção de temperatura oferecida pelo isolante para os dispositivos de captura de dados, foi efetuado um teste que põe à prova as suas capacidades de reflexão da radiação, bem como a capacidade da comunicação na rede ZigBee quando exposta a altas temperaturas.

Descrição: Foram expostos dois dispositivos de captura de dados a uma fonte de calor, originada com a combustão de gás propano: um com apenas uma camada de *fire shelter*, e outro contendo uma segunda camada de revestimento de modo a isolar também os sensores, como se mostra na Figura 5.3.Isto permite medir as temperaturas de exposição à chama com e sem a proteção do *fire shelter*.

A chama foi lançada de uma distância de aproximadamente 50 cm dos dispositivos emissores de dados. Esta foi originada na linha recta entre os dispositivos emissores e o recetor, que se encontrava a uma distância de 10 metros. A chama foi lançada diretamente aos dispositivos emissores. No entanto, e devido ao facto de o gás estar em débito reduzido, esta mesma chama tem a tendência de subir. Isto leva a que a chama não tenha atingido diretamente os dispositivos, mas estando muito perto dos mesmos.

Os dispositivos estão também equipados com placas de rede ZigBee para que se consegua obter na estação base as temperaturas de cada um dos dispositivos ao longo do tempo.

Resultados

Na Figura 5.4 são apresentadas as várias leituras de ambos os sensores ao longo do tempo. Nas Figuras 5.5 e 5.6 podemos observar a variação da temperatura em cada um dos sensores em separado. Para ajudar a interpretar os valores e defender uma posição relativamente ao isolante escolhido, é apresentada a Tabela F.2 com um breve resumo dos valores obtidos nos dois sensores.

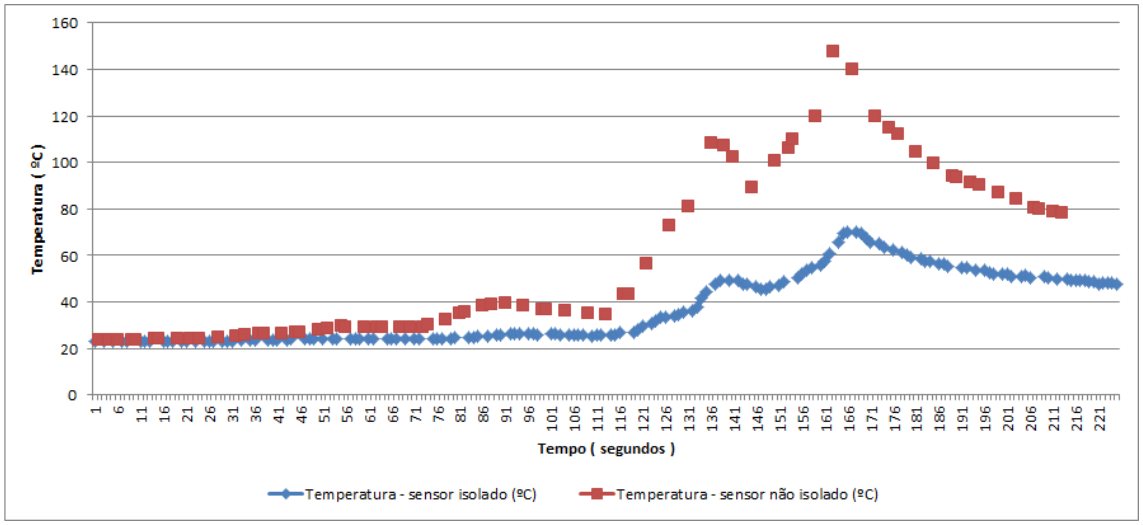


Figura 5.4: Temperaturas recebidas ao longo do tempo

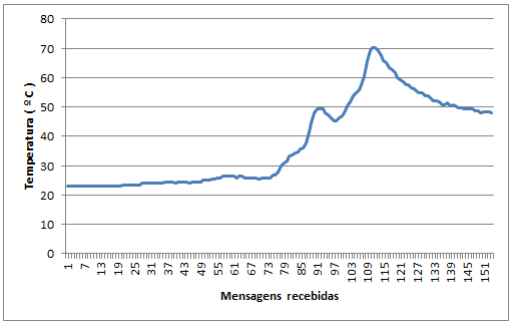


Figura 5.5: Temperaturas lidas pelo sensor não isolado

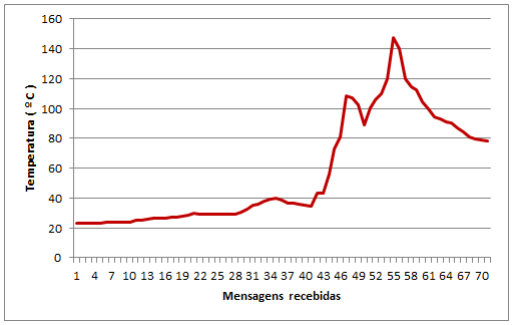


Figura 5.6: Temperaturas lidas pelo sensor não isolado

Figura 5.7: Gráficos de teste da temperatua dentro e fora do *fire shelter*

Relativamente às leituras feitas no presente teste, foram recebidos mais dados de um dispositivo do que do outro, embora estivessem programados para enviar mensagens com a mesma frequência. Pode considerar-se que esta leitura se deve a perda de mensagens na rede ou ineficiência por parte do recetor na receção dos dados.

Análise/Conclusões

A partir dos gráficos da Figura 5.7 é possível observar as diferenças de temperatura entre o sensor exposto diretamente à combustão e aquele que está protegido pelo *fire shelter*. É notório que a partir do momento em que a temperatura começa a subir, o sensor

não isolado revela sempre uma temperatura superior. Essa diferença de temperatura é ainda mais acentuada nos picos de calor.

Estes valores podem ser observados pormenorizadamente na Tabela F.2. Desta forma conseguiu-se provar que o *fire shelter* é um bom isolante para proteger os componentes eletrónicos da temperatura a que possam estar expostos. Por outro lado, pode ainda concluir-se que a temperatura não afeta acentuadamente a comunicação da rede ZigBee, pois conseguiram-se obter dados durante todo o periodo de teste, inclusivé nos picos de temperatura.

5.3 Teste dos dispositivos em ambiente de temperatura elevada

Figura 5.8: Teste de exposição de disposivos a temperatura elevada

Para terminar o conjunto de testes que dizem respeito à resistência dos dispositivos e às capacidades de comunicação dos mesmos em temperaturas elevadas, foi efetuado um terceiro teste onde dois dispositivos foram submetidos a um ambiente fechado com temperatura elevada. O seu objetivo foi provar a resistência dos dispositivos desenvolvidos bem como garantir a capacidade de comunicação da rede ZigBee quando o emissor está fechado num ambiente quente.

Descrição: Foram colocados os dois dispositivos desenvolvidos no interior da caixa apresentada na Figura 5.8, e posteriormente fechados o orifício superior e a porta lateral. Através do orifício do canto inferior esquerdo foi aplicada uma chama proveniente de gás propano. Os dispositivos no interior recolheram ao longo do tempo a temperatura ambiente dentro da caixa e enviaram essa informação para um recetor instalado no exterior da caixa. Ambos os dispositivos se encontravam com os sensores a descoberto. Ou seja, nenhum continha uma segunda camada de isolamento a tapar os sensores, como acontecia

no teste anterior. O recetor encontrava-se a uma distância de aproximadamente 10 metros da caixa.

Resultados

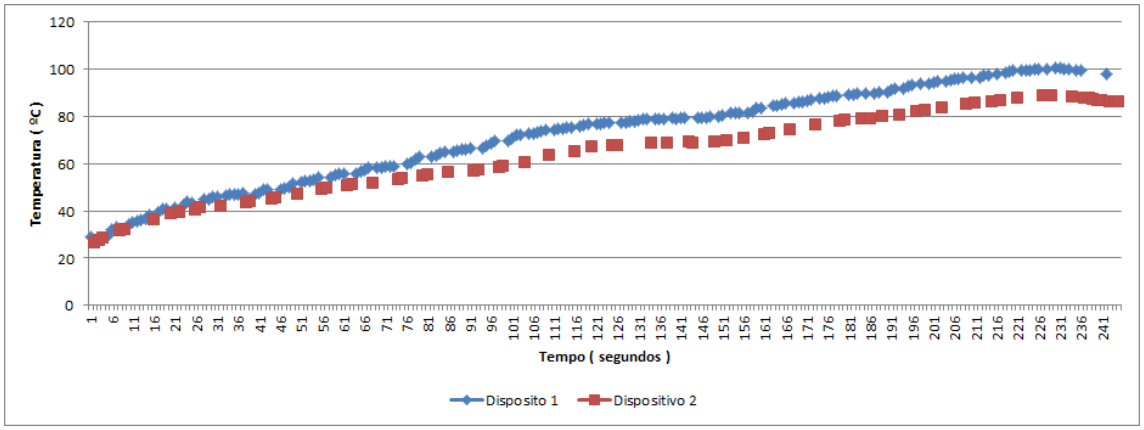


Figura 5.9: Gráfico de resultados de exposição de disposivos a temperatura elevada

No gráfico da Figura 5.9 são apresentadas as temperaturas recebidas no dispositivo recetor ao longo do tempo.

Análise/Conclusões

Durante este teste os dispositivos foram submetidos até uma temperatura máxima de 100ºC lida pelo "Dispositivo 1", altura em que foi retirada a fonte de calor. Contudo, os dois dispositivos apresentaram uma diferença nas temperaturas lidas, possivelmente por uma estar mais proxima do local onde foi aplicada a combustão.

Após este teste os dispositivos foram retirados da caixa, ainda em funcionamento. Pode assim concluir-se que os dispositivos resistem fechados num ambiente com temperaturas elevadas, e que ainda assim a comunicação da rede não é afetada com o aumento da temperatura ambiente.

5.4 Testes aos serviços

Para testar o desempenho do *middleware* desenvolvido para a estação base, foi efetuado um conjunto de testes de carga aos serviços na *Web* que este disponibiliza, os quais são apresentados de seguida. Estes testes aos serviços foram feitos com o auxílio da aplicação SOAP UI, a qual possibilita o envio correto de pedidos SOAP após análise do ficheiro WSDL dos serviços.

Decidiu-se que para avaliar o desempenho do *middleware*, o melhor serviço seria o "readObservation", pois é aquele que tipicamente exige um maior processamento devido às operações que necessita efetuar. O funcionamento deste serviço consiste em ler os dados do disco, filtrá-los de acordo com o pedido enviado e criar o ficheiro XML com os dados resultantes para enviar ao cliente. Outro fator que levou a eleger este serviço para

os testes de desempenho é este ser um dos mais críticos e vitais para o desempenho do sistema final.

Para efetuar estes testes recorreu-se a um *gateway* instalado no *middleware* que permite receber mensagens no mesmo formato que as enviadas pelos dispositivos de captura de dados, mas aqui o meio de comunicação será via *sockets* TCP/IP. Através de uma aplicação de testes desenvolvida em Java é possível enviar, através da rede *wireless*, as mensagens de simulação de observações.

5.4.1 Teste de leitura de um número fixo de observações

Este teste tem como objetivo analisar o comportamento do *software* que fornece os serviços quando são efetuados vários pedidos iguais, situação em que deve fornecer sempre o mesmo conjunto de observações.

Descrição: Foi criada uma ocorrência no sistema e associadas 20 observações. Posteriormente foram feitos 25 pedidos ao serviço, requisitando as 20 observações associadas a essa ocorrência. As 20 observações estão associadas a 2 bombeiros (10 a cada), e cada uma contém valores relativos a temperatura, monóxido de carbono e dióxido de azoto.

Resultados

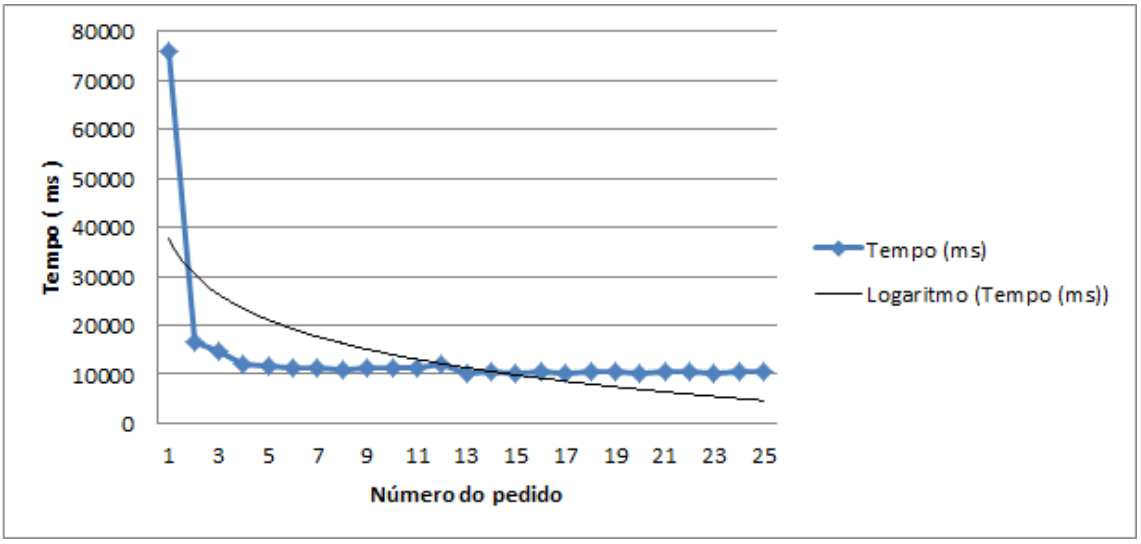


Figura 5.10: Teste de serviços com um número fixo de observações

No gráfico da Figura 5.10 são mostrados os tempos de cada um dos pedidos. A ideia da linha logaritmo apresentada em conjunto com os valores lidos, é mostrar a tendência dos valores do tempo em relação ao número de pedidos, a criar uma função logarítmica. Todos os dados podem ser confirmados na tabela apresentada no Apêndice F.3, onde podemos ver os valores exatos, em milisegundos, de cada um dos pedidos.

Análise/Conclusões

Como podemos observar no gráfico, o primeiro pedido demora imenso tempo a ser processado, notando-se uma descida acentuada entre os tempos de processamento do primeiro e segundo pedidos. Em contrapartida, a partir do segundo pedido não existe quase diferença no tempo que estes demoram a ser processados. A partir do 13º pedido não se nota mesmo qualquer oscilação nos valores, dado que a diferença entre cada par de valores consecutivos nunca atinge os 1000 milisegundos.

A grande diferença entre o primeiro e o segundo pedido tem a ver com as leituras a partir do disco e alocação de memória. Ou seja, a primeira vez que o pedido é efetuado não existem dados em memória, todos estão guardados na base de dados (disco). Os processos de leitura/escrita no disco são sempre mais demorados que outros cálculos efetuados e acessos à memória, daí esta discrepância de desempenho, uma vez que no primeiro pedido todos os dados são lidos do disco e alocados na memória demorando bastante tempo. Consequentemente, no segundo pedido já todos os dados necessários se encontram em memória.

Habitualmente, nestes testes, é sempre necessário um "aquecimento". Por exemplo, o teste é executado 40 vezes e apenas são contadas metade das execuções, descartando as 10 primeiras e as 10 últimas. O resultado é obtido a partir da média dos 20 resultados considerados. Contudo, não realizámos esse "aquecimento" para dar a entender uma ideia global do que acontece aos sistema.

5.4.1.1 Teste de variação do número de observações

De modo a analisar a forma como o número de observações a processar pelo *middleware* afeta o tempo de espera da aplicação cliente, foi feito um teste onde o número de observações foi aumentando após cada pedido.

Descrição: Foram feitos 25 pedidos de observações a uma determinada ocorrência à qual estão associados dois bombeiros. Em cada um desses pedidos existe uma diferença de duas observações, ou seja, no momento do primeiro pedido existiam duas observações e estas foram aumentando gradualmente, adicionando duas observações após cada pedido. Previamente foi requisitado um conjunto de serviços para garantir que os dados pretendidos, à exceção das observações, já estavam em memória quando o teste se iniciasse. Assim, podemos observar apenas o impacto do número de observações, sem perdas de tempo nos primeiros pedidos a reservar outros dados necessários em memória. Os serviços requisitados foram: "alterDateTime" (380 milisegundos); "getCapabilities" (8228 milisegundos); "addOffering" (5031 milisegundos). Complementarmente tinha ainda sido requisitado o serviço "readObservation" várias vezes de modo a efetuar o "aquecimento" da máquina virtual do Java.

Resultados

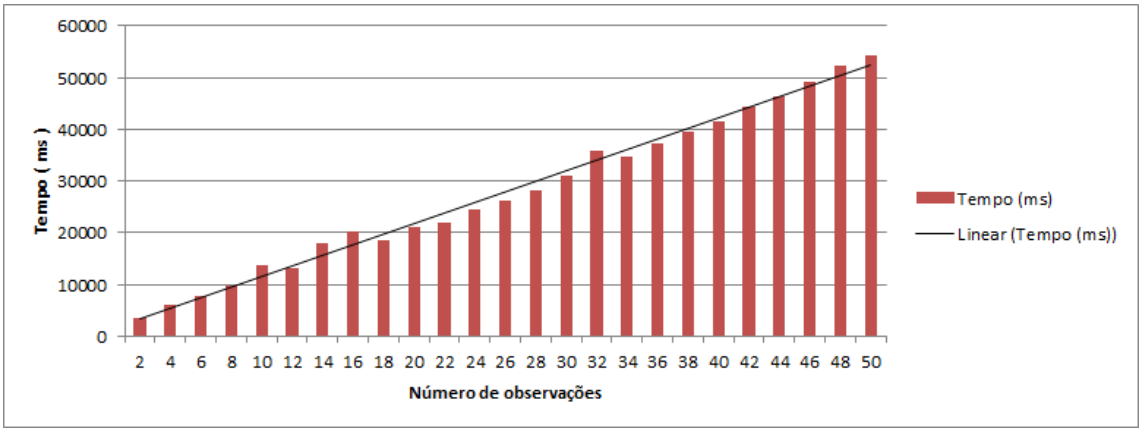


Figura 5.11: Teste de serviços com variação do número de observações

No gráfico apresentado na Figura 5.11 podemos observar o tempo de resposta (em milisegundos) a cada um dos pedidos efetuados ao serviço "readObservation". Em relação à linha com o nome "Linear", apresentada no gráfico, esta tem como objetivo mostrar uma tendência para o aumento linear do tempo de resposta à medida que aumentam as observações.

Análise/Conclusões

Neste gráfico, através da tendência linear verificada, pode concluir-se que o aumento do número de observações pedidas ao *middleware* origina um aumento linear no tempo de processamento dos dados a enviar. Daí podemos afirmar que esta operação tem uma complexidade O(n).

5.4.2 Teste de carga de utilizadores

Dadas as limitações em termos de *hardware* dos dispositivos onde é executado o *middleware*, o sistema fica ainda mais exposto e vulnerável, ainda que este esteja limitado a um máximo de 50 utilizadores ligados à rede *WiFi*. Daí surge a necessidade de avaliar a sua fiabilidade e robustez relativamente à quantidade de pedidos e ao número de utilizadores dos seus serviços.

Descrição: Foi criada uma ocorrência e adicionadas 20 observações. Para simular um aumento na carga de utilizadores, foi criado um conjunto de testes na aplicação SOAP UI, à qual se adicionou o pedido de leitura de observações. Em seguida foram feitos vários testes até que o sistema deixasse de responder e enviasse mensagens de erro para os ficheiros de *log*. Após cada teste bem sucedido aumentou-se o número de *threads* a fazer pedidos em simutâneo do mesmo pedido, de modo a simular um maior número de utilizadores.

Antes de iniciar o teste, foram feitos vários pedidos de observações associadas à ocorrência pretendida. A finalidade foi alocar previamente os dados em memória, para

garantir que os tempos obtidos não sofrem influência de operações de leitura a partir do disco. Assim, garante-se uma análise exclusiva à influência do aumento do número de utilizadores em simultâneo.

Resultados

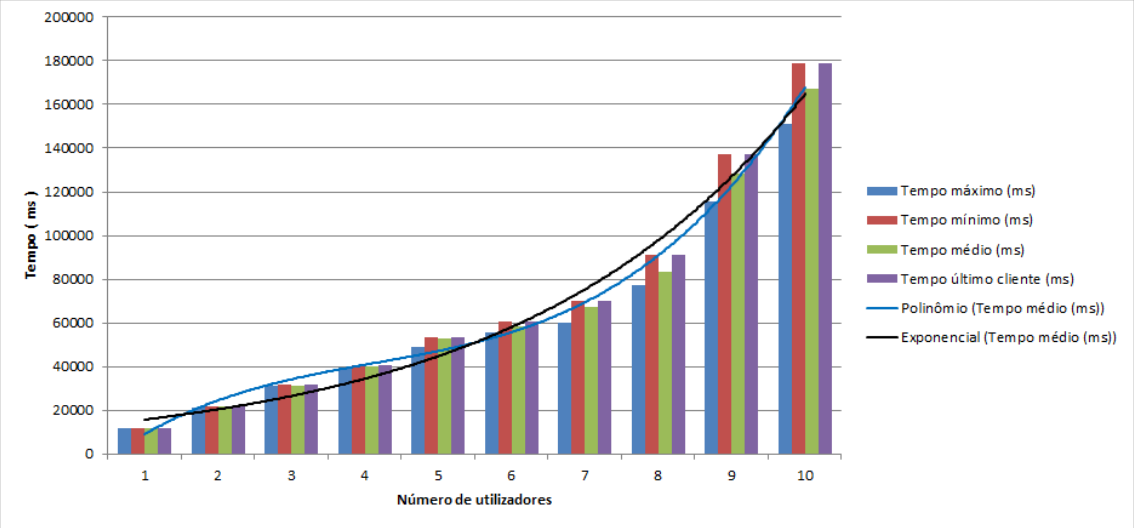


Figura 5.12: Teste de serviços com aumento do número de utilizadores

Os dados referentes a este teste podem ser analisados mais detalhadamente na Tabela do Apêndice F.5.

Relativamente ao gráfico da Figura 5.12, existem duas linhas, calculando duas possíveis tendências para o gráfico: crescimento exponencial ou polinomial de ordem 3.

Todos os dados relativos aos tempos apresentados na tabela e no gráfico acima referidos estão em milisegundos. O eixo das abcissas representa o número de utilizadores e consequentemente as etapas do teste.

Análise/Conclusões

Tal como pode ser observado no gráfico, não existe grande discrepância entre os tempos máximo e mínimo em cada uma das etapas do teste. Contudo, podemos ainda verificar que à medida que o número de utilizadores aumenta, essa diferença torna-se um pouco mais acentuada. Através da tabela é possível ainda observar que, tipicamente, o último pedido a ser efetuado é aquele que obtém o maior tempo contabilizado desde o momento em que é feito até se obter uma resposta.

Na 11ª etapa do teste obteve-se um erro de memória (*OutOfMemory*) e foi excedido em cada uma das *threads* o *timeout* definido para os pedidos (300 000 milisegundos), pelo que não se prosseguiu com o teste. Foi feita uma segunda tentativa para confirmar estes resultados, que revelaram uma incapacidade por parte da estação base para atender a vários utilizadores em simultâneo. Esta conclusão foi obtida também tendo em conta a ausência de alguns fatores de processamento, tais como a alocação de dados em memória e números elevados de observações. Ou seja, mesmo na ausência de alguns fa-

tores que prejudicariam o processamento das respostas aos pedidos, estes revelaram ainda um tempo de processamento demasiado elevado em relação ao que era desejado.

O baixo número de dados obtidos, devido à incapacidade do dispositivo para atender a vários utilizadores, não permitiu analisar em concreto qual a tendência do crescimento do tempo de processamento relativamente ao aumento do número de utilizadores. Desta forma, foram tidas em conta duas possíveis linhas de tendência para o aumento do tempo. Estas linhas foram calculadas de acordo com os tempos médios em cada uma das etapas. De acordo com a aplicação Microsoft Excel estas linhas apresentam as seguintes equações: $y = 12158e^{0.2606x}$ para o crescimento exponencial; $y = 450.33x^3 - 5577.9x^2 + 29008x - 14750$ para o crescimento polinomial.

No caso de se assumir um crescimento polinomial utilizando a tendência calculada pela aplicação, e calculando os seus pontos de inflexão, podemos obter um número de utilizadores até ao qual os tempos de processamento não aumentam significativamente. Os pontos de inflexão apontam os momentos a partir dos quais o gráfico da função altera a sua concavidade. Estes podem ser calculados a partir da segunda derivada da função, bastando encontrar os zeros dessa derivada. Assim, tem-se:
$y = 450.33x^3 - 5577.9x^2 + 29008x - 14750$,
$y' = 29008 - 11155.8x + 1350.99x^2$,
$y'' = -11155.8 + 2701.98x$.
E por último calcula-se:
$y'' = 0 \Leftrightarrow -11155.8 + 2701.98x = 0 \Leftrightarrow 2701.98x = 11155.8 \Leftrightarrow x = \frac{11155.8}{2701.98} \Leftrightarrow x \approx 4.12875$.
Obtém-se um zero da função no intervalo $]4; 5[$. Assim, temos o ponto de inflexão do gráfico da função polinomial entre as abcissas 4 e 5. Desta forma podemos concluir, e confirmar por inspeção do gráfico, que os aumentos mais significativos nos tempos de processamento se verificam a partir dos 5 utilizadores em simultâneo, inclusivé.

5.5 Teste de sistema

A melhor avaliação do funcionamento e das capacidades de um sistema é colocá-lo em funcionamento num ambiente real. Assim, decidiu-se utilizar o sistema em ambiente de incêndios florestais, de maneira a avaliar os dados a que os bombeiros estão expostos. Para tal, os dispositivos foram colocados no veículo de combate a incêndios florestais dos Bombeiros Voluntários de Vendas Novas nos dias que me encontrei de serviço na Equipa de Combate a Icêndios (ECIN) da mesma corporação, que foram de 5 a 8; 11 e 12; de 18 a 20; 26 e 27 de Agosto de 2014.

Descrição: A estação base foi colocada dentro do veículo atribuído à equipa alimentado através de uma porta USB que este dispõe para ligação de aparelhos eletrónicos. A aplicação móvel foi instalada no meu *smartphone* (Samsung Galaxy S2 Duos), para que

a cada ocorrência a que me deslocasse, conseguisse criar uma nova ocorrência no sistema, de modo a que este possa guardar os dados enviados pelos dispositivos móveis. Em termos de dispositivos de captura de dados, apenas foi utilizado um.

Resultados

Infelizmente para o projeto mas felizmente para o país, apenas me desloquei a uma ocorrência que necessitasse de fazer combate a incêndio. Do ponto de vista do projeto, a falta de ocorrências pode ser considerada um problema, na medida em que foram recolhidos poucos dados para avaliar o funcionamento do sistema e a exposição dos bombeiros aos fatores de risco.

Os dados recolhidos da única ocorrência são apresentados na Figura 5.13, a qual é um *print screen* da página de "Monitorização" da aplicação.

O incêndio foi de pequenas dimensões, onde ardeu essencialmente pasto e onde, tal como podemos observar na figura, a temperatura foi pouco além dos 30ºC.

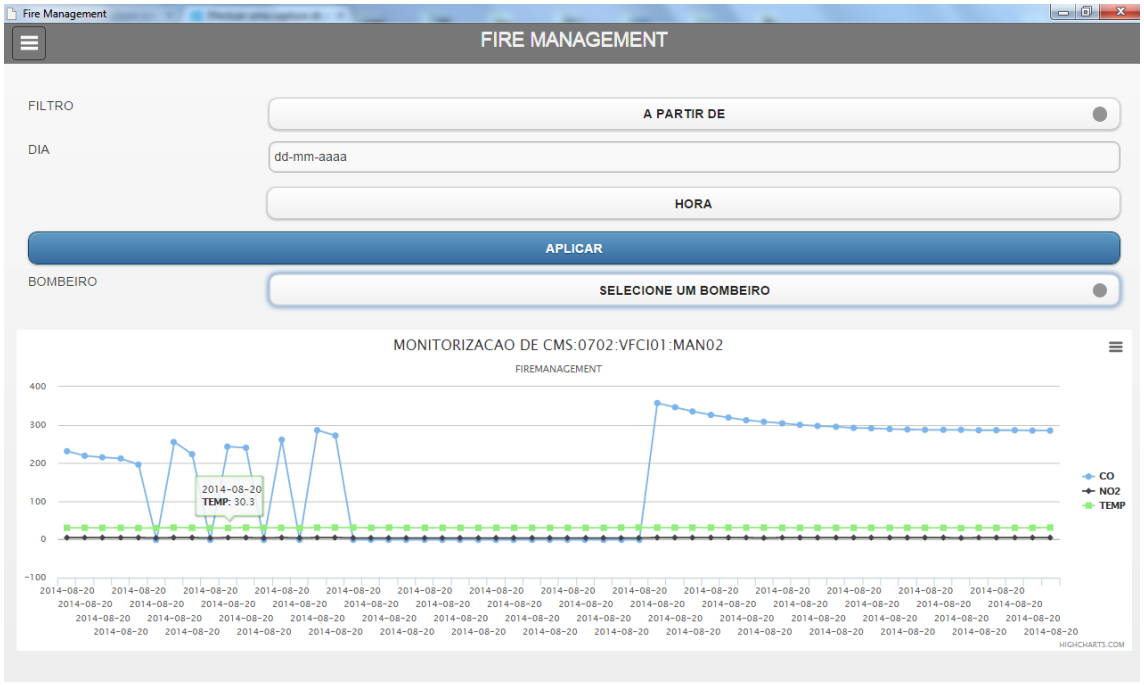


Figura 5.13: Gráfico de variação dos dados recolhidos durante um incêndio rural

Análise/Conclusões De acordo com os dados recolhidos e apresentados no gráfico da Figura 5.13, verifica-se que a temperatura sofreu variações pouco significativas, assim como o dióxido de azoto, que demonstra valores quase nulos. O monóxido de carbono apresenta valores mais significativos. O sensor pode assumir valores entre 0 e 1023, e este apresenta oscilações entre os 200 e 400. Estes valores não foram convertidos para uma unidade de medida de gases, pelo que apresentam os valores analógicos lidos diretamente do sensor. Quando os valores do monóxido de carbono são mais altos, significa que havia maiores concentrações desse gás no local. Relativamente às quebras mais significativas

dos valores deste gás para perto de 0 verificadas no gráfico, estão relacionadas com o período de limpeza do sensor (*cf.* Secção 4.1.1.1.2).

Cada ponto do gráfico representa uma observação recebida pelo *middleware* que deve ficar registada com a hora de receção. No entanto, houve um problema com a aplicação que fez com que fosse mostrada a data e não a hora.

O período de limpeza do sensor é de 90 segundos e o de medição é de 60 segundos e o dispositivo envia observações para o *middleware* aproximadamente a cada segundo. No entanto, o gráfico apresenta, em determinadas alturas, apenas uma ou duas medições até nova limpeza do sensor, ou apenas uma observação durante o período de limpeza até voltar a medir valores do ambiente. Isto pode dever-se a problemas da rede ZigBee ou à fraco desempenho do dispositivo da estação base enquanto receptor de dados.

Capítulo 6

Conclusão

Hoje em dia podemos observar redes de sensores nas mais variadas áreas, e este projeto consistiu na aplicação a uma nova àrea, onde até então pouco se ouviu falar. Contudo, o paradigma da Internet das Coisas vai um pouco além das redes de sensores. O que se pretendeu explorar, além da aplicação de uma rede de sensores em ambiente adverso, foi também a possibilidade de combinar um vasto leque de tecnologias para atingir um determinado fim: a monitorização das condições a que os bombeiros estão expostos em situação de combate a incêndios florestais.

Neste projeto levei a cabo a contrução e o desenvolvimento de um sistema composto por: (1) uma rede de sensores sob protocolo de comunicação ZigBee, constituída por dispositivos móveis de captura de dados; (2) um *middleware* escalável, extensível e portável capaz de integrar redes de sensores e disponibilizar serviços na *Web*; (3) uma aplicação cliente multiplataforma que pode ser tanto executada em computadores através de um *browser*, como instalada em *smartphones*, em conformidade com as características de uma aplicação nativa.

Os cenários de incêndios florestais são, grande parte das vezes, bastante adversos ao uso de equipamentos eletrónicos, principalmente devido às temperaturas a que estão expostos. A maior parte dos equipamentos eletrónicos comuns são sensíveis a temperauras que excedam os 50ºC. Contudo, quando devidamente protegidos da temperatura, estes podem entrar em ambientes que excedam as suas temperaturas limite. Assim, é possível criar uma rede de sensores com dispositivos ditos comuns, para serem usados em ambientes hostis, sem haver a necessidade de desenvolver componentes próprios ou adquirir alguns mais robustos, os quais podem ser muito dispendiosos. Este facto foi comprovado durante o teste apresentado na Secção 5.3, pois o ambiente onde os dispositivos estavam, encontrava-se a cerca de 100ºC, e estes continuavam em funcionamento. A explicação para isto pode também ser dada pelo teste da Secção 5.2, onde são apresentadas as diferenças de temperatura dentro e fora do isolante utilizado.

Esses dispositivos eletrónicos, revelaram-se um grande entrave ao desenvolvimento do projeto, principalmente devido às minhas fracas bases de eletrónica. Foi necessário

tempo extra, além do previsto, para entender o funcionamento dos sensores adquiridos e para a construção dos dispositivos para captura dos dados. Isto teve, como é óbvio, implicações no desenvolvimento das restantes partes do sistema. Foi necessário acelerar o desenvolvimento do *middleware*, estação base e aplicação cliente, deixando para trás alguns pormenores de otimização do desempenho. Outra consequência desses atrasos foi a redução do tempo reservado para testar o sistema.

Além do pouco tempo de que dispus para testar o sistema, surgiu ainda outro problema impossível de contornar, que foi o baixo número de ocorrências de incêndios florestais no período que este esteve sujeito a teste. Foi apenas efetuado um teste ao sistema em cenário real, o que deitou por terra a tentativa de angariar um elevado conjunto de dados em ambientes aos quais o sistema se destina. Contudo, a única ocorrência em que foi possível testar o sistema, o resultado foi positivo, tendo conseguido registar, por exemplo, temperatura ambiente, a qual não contou com valores desproporcionados. Em relação aos gases notou-se ainda variações dos valores de monóxido de carbono. Infelizmente, não foi possível efetuar a conversão dos valores obtidos nos sensores de gases para ppm's, a unidade de referência para quantificar gases poucos abundantes. Portanto, a análise de valores dos gases, foi feita apenas de acordo com as leituras directas à voltagem de saída do sensor, as quais são convertidas pelo microcontrolador, para um número inteiro entre 0 e 1023 inclusivé.

Podemos então concluir, como análise final, que o sistema funciona, dentro daquilo que lhe foi inicialmente proposto, embora com alguma limitações no que se refere ao desempenho do equipamento. Isto porque estas limitações se baseiam na capacidade de processamento do dispositivo da estação base.

Como trabalho futuro, pretende-se efetuar melhorias ao nível do *middleware*, e também ao nível do *hardware* que o compõe, podendo este ser alterado futuramente por outro dispositivo de computação miniaturizada.

Pretende-se também proceder a um conjunto de testes em cenário real e recolher informação sobre as condições a que os bombeiros estão expostos. Estudo deste tipo para fogos florestais têm sido pouco explorados e há interesse por parte das autoridades em ter este tipo de informação.

Para concluir, pensamos submeter um pedido de utilidade sobre o sistema desenhado ao Instituto Nacional da Propriedade Industrial (INPI), a fim de proceder à sua comercialização.

Apêndice A

Padrão das mensagens de comunicação na rede de sensores

Estas mensagens são formadas por uma cadeia de caracteres com os seguite formato:
ENDPOINT:cms:id_cb:id_veiculo:id_bombeiro;TEMP:temperatura;CO:valor_co;NO2:valor_no2;LAT:laitutde;LNG:longitude

- "ENDPOINT" - indica o dispositivo que envia os dados. Dentro deste campo são apresentadas três informações, sendo elas:
  - cms: valor estático, que finciona como um identificador de objetos do sistema.
  - id_cb: identificação do corpo de bombeiros. Cada corpo de bombeiros em Portugal tem um número único de identificação composto por quatro dígitos, sendo que os dois primeiros indicam o distrito e os dois ultimos o número do corpo de bombeiros dentro desse distrito.
  - id_veiculo: identificação do veículo dentro do corpo de bombeiros. Tipicamente composto por seis caracteres. Os primeiros quatro indicam a tipificação do veículo (o tipo de veículo, a que serviço se destina). Os últimos dois caracteres são o número do veículo desse tipo dentro da corporação a que pertence.
  - id_bombeiro: valor meramente informativo do bombeiro que transporta o dispositivo. Não podem existir dois valores iguais no mesmo veículo. Normalmente são utilizados neste campo os valores "MAN01", "MAN02", "MAN03", etc.
- "TEMP" - refere a temperatura ambiente expressa em ^{o}C.
- "CO" - apresenta o valor do monóxido de carbono, representado por um valor inteiro lido diretamente do sensor através de portas analógicas.
- "NO2" - mostra o valor do dióxido de azoto, representado por um valor inteiro lido diretamente do sensor através de portas analógicas.

- “LAT” - latitude do local do dispositivo.
- “LNG” - longitude do local do dispositivo

Apêndice B

Dispositivo de captura de dados

B.1 Placa principal

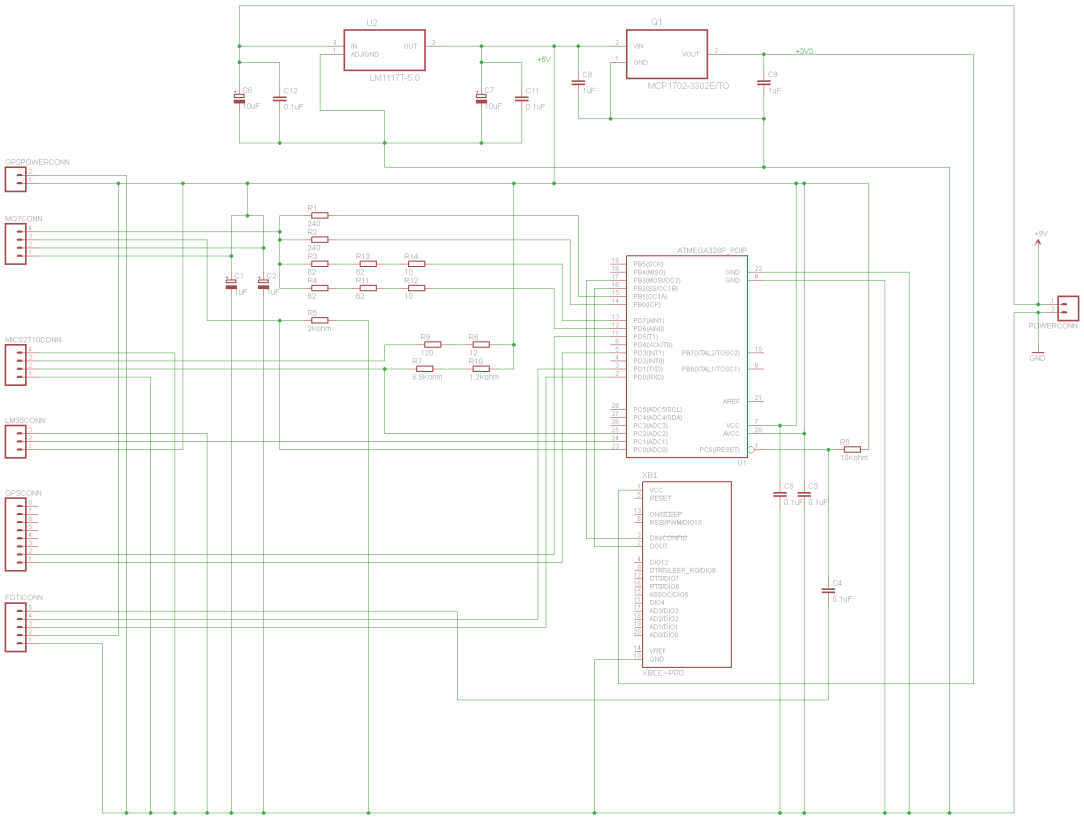


Figura B.1: Esquema do circuito do dispositivo móvel

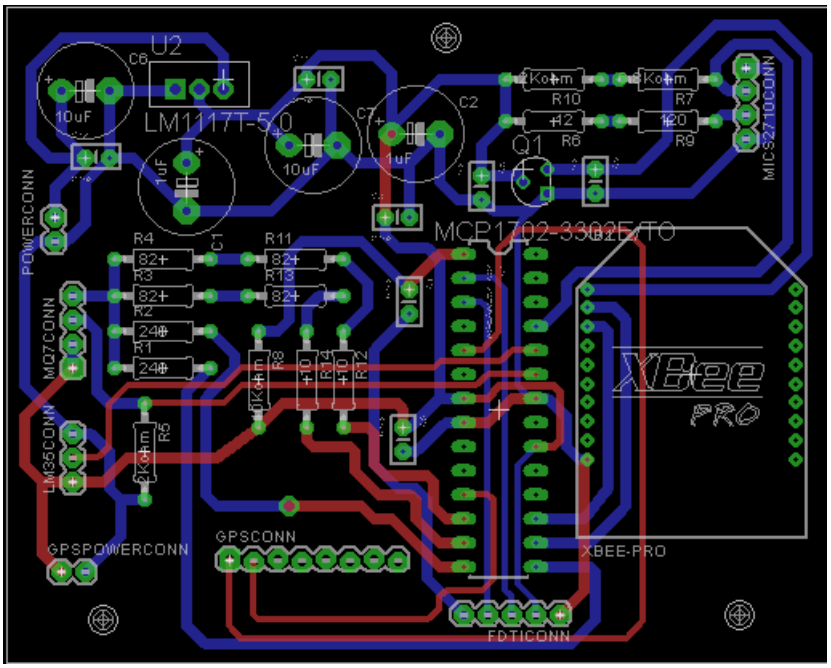


Figura B.2: Desenho da PCB para o dispositivo móvel

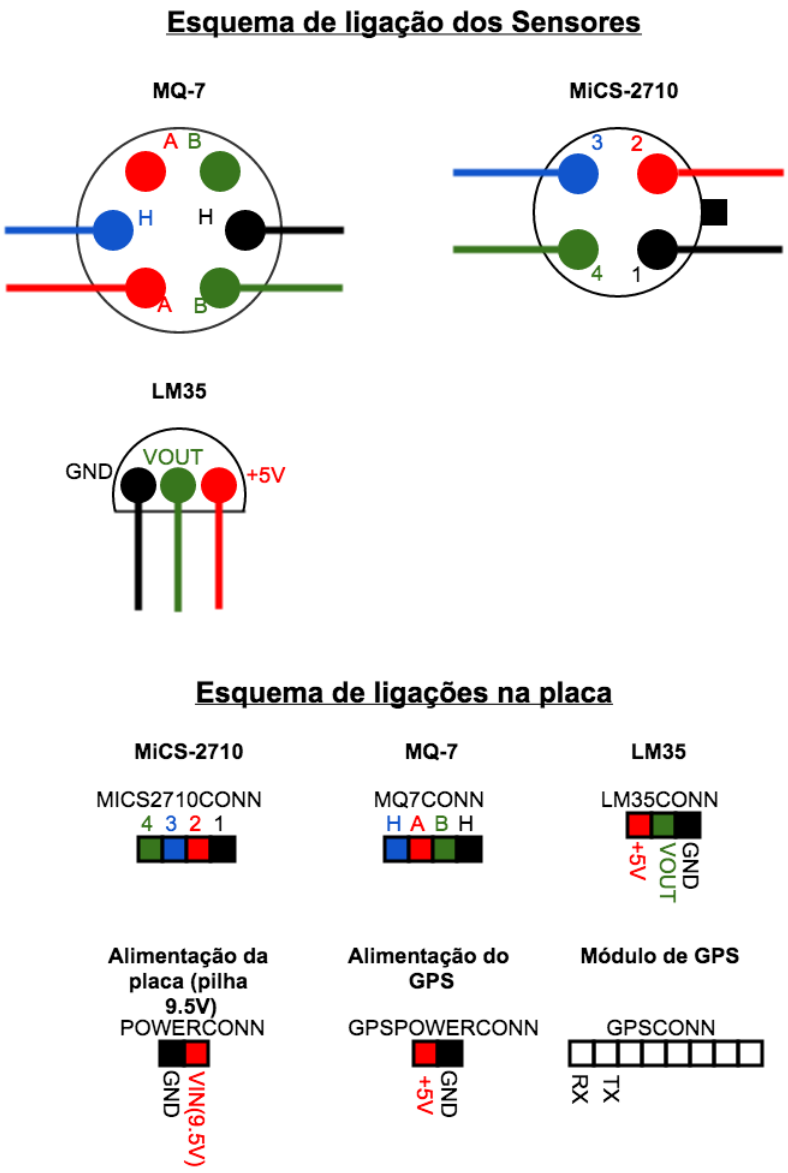


Figura B.3: Esquema de ligação de sensores e pinos da PCB

B.2 Microcontrolador

B.2.1 Instalação de *Bootloader*

Para se poder programar um novo microcontrolador Atmega328, o usado neste projeto, é necessário instalar primeiro o *bootloarder*. A forma mais fácil de o fazer é recorrer a um Aruino que esteja configurado como ISP de programação.

Para transformar um Arduino num ISP de programação basta ligar a placa Arduino através de USB, e fazer *upload* do programa ArduinoISP, disponível nos exemplos que acompanham a aplicação ArduinoIDE. Posteriormente esfetua-se a ligação do circuito representado na Figura B.4, recorrendo, por exemplo, a uma *breadboard* para colocar o microcontrolador, os condensador e a resistência necessária.

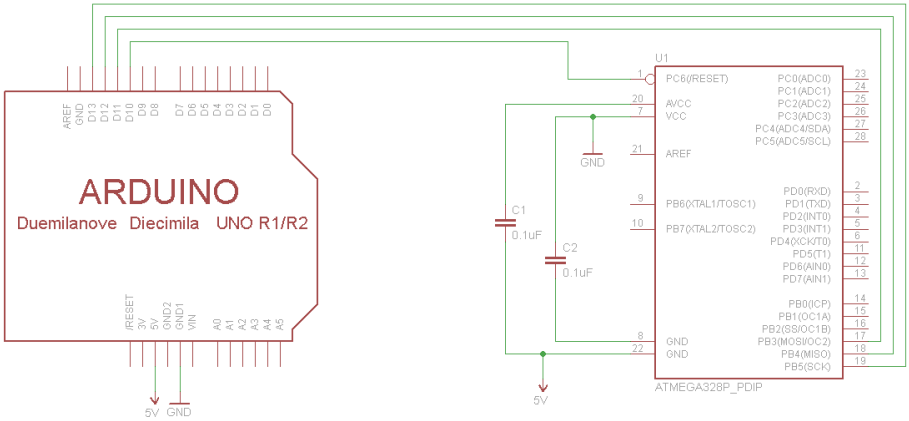


Figura B.4: Esquema do circuito para instalação do *bootloader*

Após efetuar as ligações no menu (“Tools” > “Board”) escolhemos o dispositivo “ATmega328 on a breadboard (8 MHz internal clock)”, necessitando previamente de adicionar ao ArduinoIDE as suas especificações, que podem ser obtidas no *site* do Arduino [3].

Por último, basta selecionar o modo de programação com Arduino (“Tools” > “Programmer” > “Arduino as ISP”) e executar a operação “Burn bootloader” através do menu (“Tools” > “Burn Bootloader”).

B.2.2 Carregamento de programas

Após a isntalação do *bootloader* podemos fazer *upload* de programas para o microcontrolador ainda através do Arduino. Contudo, agora há que alterar o circuito de acordo com a Figura B.5, e retirar o microcontrolador existente na placa Arduino. Em relação às configurações no ArduinoIDE, estas mantêm-se iguais, bastando no final fazer upload do código normalmente.

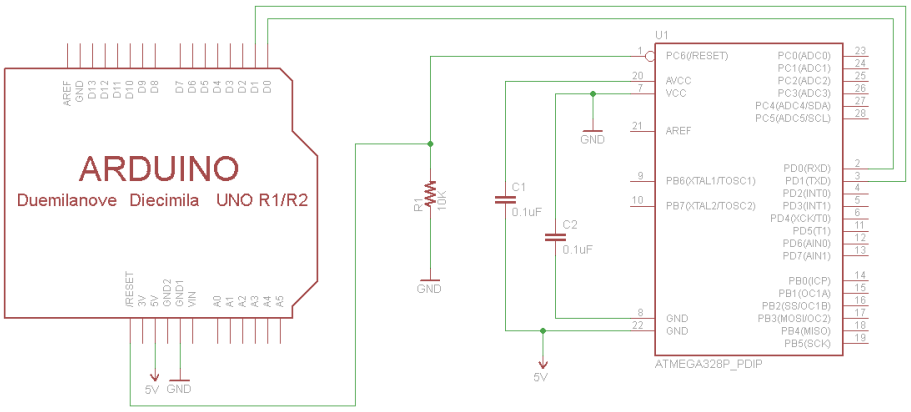


Figura B.5: Esquema do circuito para *upload* de programas para o Atmega328

Nota: A placa Arduino apresentada na Figura B.5 não deve conter o microcontrolador Atmega328.

Apêndice C

Configurações do sistema de estação base

A estação base tem como dispositivo de hospedagem o Raspberry Pi Modelo B com o sistema operativo *Raspbian*. No entanto, existem várias configurações necessárias ao sistema operativo para que o dispositivo possa cumprir os objetivos definidos.

C.1 Esquema de ficheiros e diretorias do sistema

De modo a que tudo funcione como previsto é necessário que o sistema contenha todos os ficheiros necessários nas localizações corretas. Desta forma, todos os ficheiros relativos ao funcionamente do *middleware* devem permanecer na diretoria "/home/pi/Middleware". Aqui devem constar:

- **cms.jar**: Componente principal do *middleware*. Contém todas as classes executáveis, exceto módulos adicionais e *gateways*.
- **database.sqlite** : Base de dados SQLite. Ficheiro que reúne a base de dados do sistema.
- **configuration.properties** : ficheiro de configurações do *middleware*, através do qual é possível indicar às classes Java qual a localização da base de dados, dos *gateways*, e muitos outros valores.
- **middleware.log** : Ficheiro de *logs* do sistema. Guarda informação relativa à inicialização do *middleware* e dos seus componentes, bem como de todos os erros e excepções que possam ocorrer.
- **info.log**: Guarda outras informações acerca da execução do *middleware*, como por exemplo, as mensagens recebidas da rede de sensores.
- **services**: Diretoria que reúne os módulos de serviços instalados no sistema após serem compilados.

- **gateways**: Localização onde devem permanecer os ficheiros jar dos *gateways*.

Por outro lado, como se optou por criar um serviço do sistema corresponde ao *middleware* na directoria "/usr/local/bin", devem encontrar-se os ficheiros responsáveis pelo arranque e paragem do *middleware* onde:

- **Middleware-start.sh**: Inicializa o *middleware*, garantindo anteriormente que todos os serviços e configurações foram iniciados.

- **Middleware-stop.sh**: Responsável por terminar a execução do *middleware*. Limita-se a terminar todos os processos relativos ao Java, pois não são utilizados no sistema mais processos em Java para além do *middleware*.

Ainda em relação ao serviço criado, na directoria "/etc/init.d" deve constar o ficheiro "Middleware", o qual é responsável pela gestão dos comandos disponíveis para o serviço criado. Este ficheiro recorre aos dois ficheiros descritos anteriormente para efetuar as operações "start", "stop" e "restart" do *middleware*. Os comandos indicados devem iniciar, parar e reiniciar a execução do *middleware*, respetivamente.

C.2 Criação de ponto de acesso

Após a instalação das aplicações "isc-dhcp-server" e "hostapd", é necessário configurar um conjunto de ficheiros. Estes contêm os dados do ponto de acesso bem como da respetiva rede *WiFi* a ser criada.

Respetivamente ao "isc-dhcp-server" responsável por gerir os endereços da rede é necessário:

- Alterar o ficheiro "/etc/dhcp/dhcpd.conf", de modo a ficar:

```
...
# option domain-name "example.org";
# option domain-name-servers ns1.example.org,
ns2.example.org;
...
# If this DHCP server is the official DHCP server
for the local
# network, the authoritative directive should be
uncommented.
authoritative;
...
```

  Tipicamente, este passo deve passar apenas por adicionar e remover "#" em algumas linhas de forma a comentar e retirar o comentário às configrações necessárias.

- No mesmo ficheiro há ainda que adicionar as informações para os endereços de rede, acrescentando no final do ficheiro as seguintes linhas:
  ```
  subnet 192.168.10.0 netmask 255.255.255.0 {
  range 192.168.10.50 192.168.10.100;
  option broadcast-address 192.168.10.255;
  option routers 192.168.10.1;
  default-lease-time 600;
  max-lease-time 7200;
  option domain-name "local";
  option domain-name-servers 8.8.8.8, 8.8.4.4;
  }
  ```
  criando assim uma rede onde o router contará com o enderço "192.168.10.1", e os restantes dispositvos poderão obter endereços no intervalo "192.168.10.50" a "192.168.10.100".

- Alterar o ficheiro "/etc/default/isc-dhcp-server" de modo a indicar qual a *interface* de rede à qual esta aplicação estará ligada:
  ```
  INTERFACES="wlan0"
  ```
  indicando neste caso que vamos criar a rede a partir do dispositivo ligado à *interface* "wlan0".

É também necessário adicionar algumas configurações no ficheiro "/etc/network/interfaces", que se referem às *interfaces* de rede. Neste caso alterámos apenas o que diz respeito à *interface* "wlan0", pois foi a escolhida anteriormente. As configurações relativas à mesma são:

```
iface wlan0 inet static
address 192.168.10.1
netmask 255.255.255.
```

Quanto à aplicação "hostapd", há que definir as configurações, sendo necessário:

- Criar o ficheiro "/etc/hostapd/hostapd.conf", para definir as propriedades do ponto de acesso, com o seguinte conteúdo:
  ```
  interface=wlan0
  driver=rtl871xdrv
  ssid=CMS_AP
  hw_mode=g
  channel=6
  macaddr_acl=0
  auth_algs=1
  ignore_broadcast_ssid=0
  ```

```
wpa=2
wpa_passphrase=raspberry
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Nota: o *driver* a ser utilizado deve ser o "rtl871xdrv", caso se trate de um adaptador *WiFi* Adafruit, ou "nl80211" caso contrário.

- Alterar o ficheiro "/etc/default/hostapd" para:

  ```
  ...
  DAEMON_CONF=/etc/hostapd/hostapd.conf
  ...
  ```

  passando agora a indicar qual o ficheiro que contém as propriedade do ponto de acesso.

C.3 Instalação de *drivers* RXTX para comunicação através de XBee

Uma vez configurada a placa XBee e conetada ao Raspberry Pi via USB, é necessário instalar os *drivers* de comunicação RXTX para que o *middleware* possa comunicar através desta tecnologia. Visto que o *middleware* foi desenvolvido em Java as configurações devem ser ao nível do Java e acoplamento das bibliotecas necessárias.

A instalação do mesmo é feita através do comando:
`apt-get install librxtx-java`.

As configurações do mesmo passam por:

- Criar *link* na diretoria do Java para a biblioteca RXTX da arquitectura ARM. Na diretoria "/java/jdk$< versao_do_java >$/jre/lib/ext" criar o *link* através do comando:
  `ln -s ../../../../../jni/librxtxSerial-2.2pre1.so librxtxSerial.so`.

- Criar *link* diretoria do do Java para o ficheiro jar que contém as bibliotecas RXTX para arquitetura ARM. Na diretoria "java/jdk$< versao_do_java >$/jre/lib/ext" criar o *link* através do comando:
  `ln -s ../../../../../../share/java/RXTXcomm-2.2pre2.jar RXTXcomm.jar`.

Nota: Há que ter em atenção que futuramente podem surgir novas versões e na data de criação deste documento a versão das bibliotecas RXTX era a 2.2. Assim, é possível que futuramente estes comandos exatos possam não funcionar tendo que alterá-los nas partes correspondentes à versão das bibliotecas.

Apêndice D

Middleware

D.1 Modelo de dados

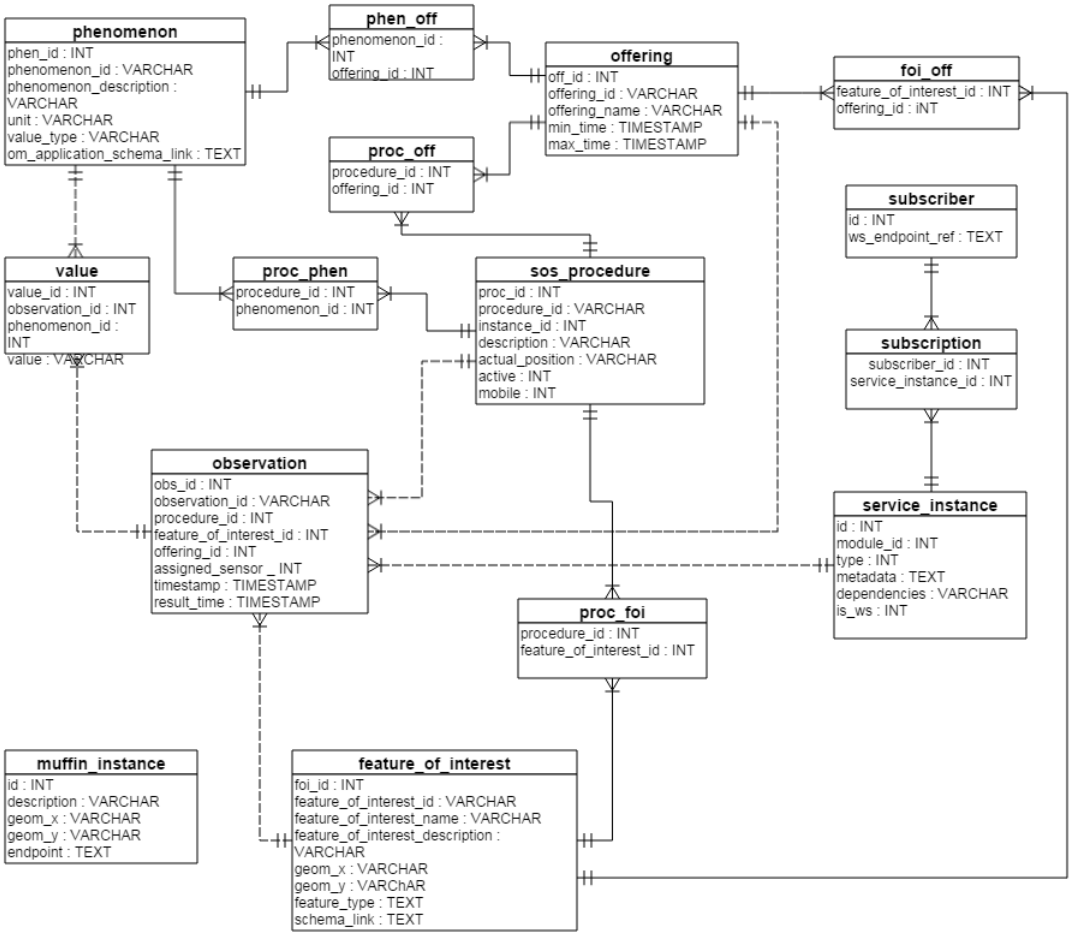


Figura D.1: Modelo de dados utilizado pelo *middleware*

D.2 Camada de acesso aos dados

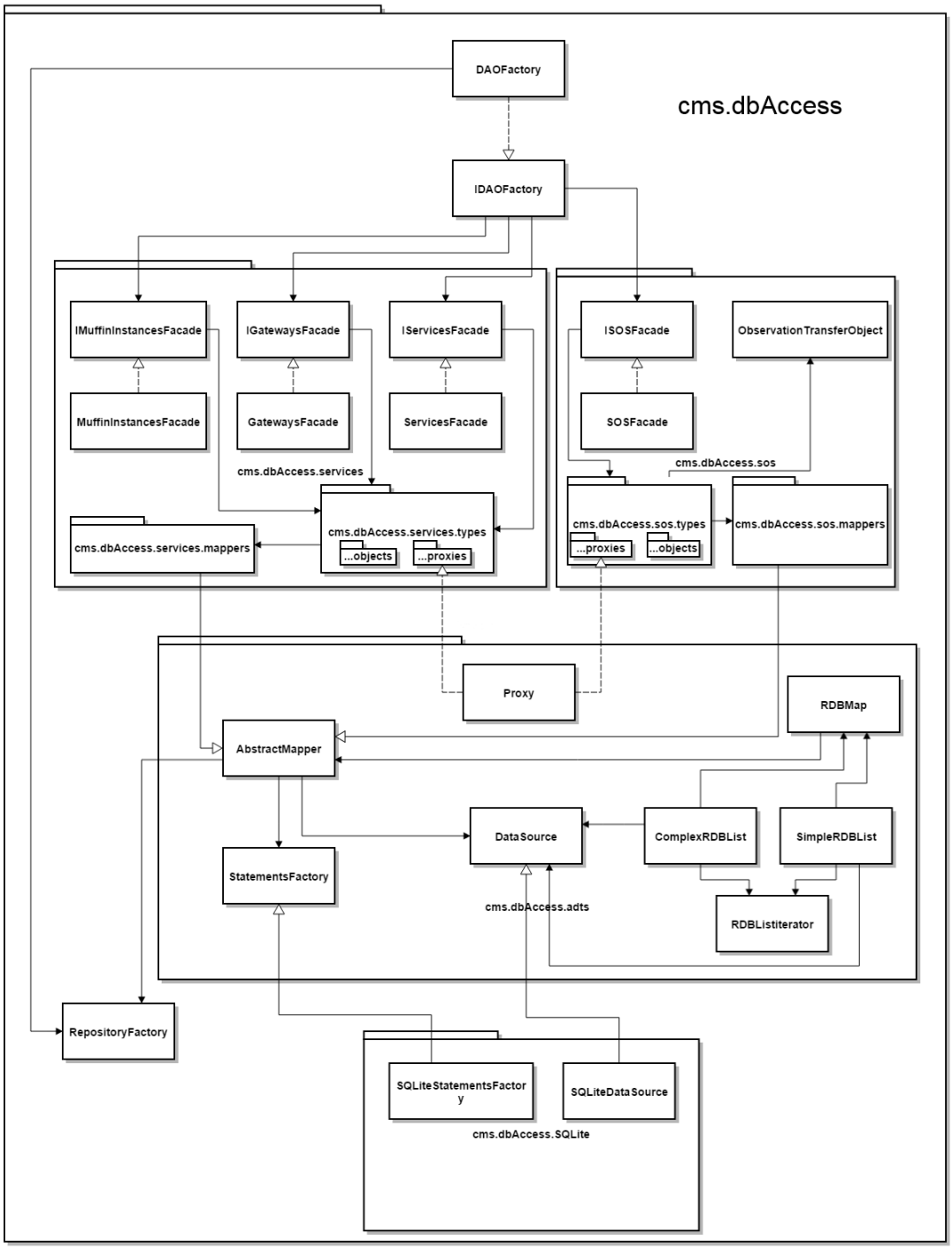


Figura D.2: Diagrama de classes e pacotes da componente desenvolvida para acesso aos dados

Apêndice E

Serviços disponibilizados

Método	Parâmetros / Resultado esperado	Descrição/Observações
addOffering	offeringId offeringName startTime endTime	Permite adicionar ao sistema uma nova offering". No contexto deste sistema em particular, uma "offering" representa uma ocorrência. Não é necessário definir de inicio os parâmetros "min_time", e "max_time" pelo que são usados para definir os limites temporais da ocorrência, definindo inicialmente apenas o "min_time" que será o tempo de inicio da ocorrência.
	"true" caso tenha sido adicionada com sucesso, "false" caso contrário.	
alterOffering	offeringId offeringName startTime endTime	Permite alterar os dados de uma "offering". Dentro do contexto do sistema atual, uma vez que uma "offering" representa uma ocorrência utiliza-se este serviço para encerrar a mesma definindo o limite temporal máximo para a mesma, ou seja, o "max_time". Apenas os parâmetros definidos causarão alterações no sistema, sendo que um parâmetro a "null" nada alterará no sistema.
	"true" caso a tenha sido alterada com sucesso, "false" caso contrário	
deployModule	instanceId moduleName description	Instala um novo módulo/serviço no sistema. Como visto anteriormente, é possível adicionar em tempo de execução novo serviços e módulos ao sistema. É através de este serviço que isso é feito, enviando o código fonte (em Java) do mesmo.
	identificador do módulo caso este tenha sido adicionado com sucesso	

getCapabilities	---	Permite apenas receber as informações sobre o sistema. Neste contexto, irá mostrar a informação sobre quais os bombeiros registados no sistema, bem como os parâmetros que podemos observar em cada um.
	conjunto de todas as "feature of interest's", "procedures" e "phenomenons" bem como as respetivas relações, em formato XML	
getModulesInfo	---	Mostra a informação de todos os módulos e gateways instalados no sistema. Indica quais deles estão instanciados (em execução) bem como as dependências entre eles.
	informação acerca dos módulos, instâncias de serviços e relações entre elas em formato XML	
inserObservation	observation	Permite inserir uma nova observação no sistema. Embora implementado este serviço não é utilizado através da interface web pois todos as inserções de observações são feitas através de um serviço próprio para isso após receber os dados oriundos do middleware.
	identificador da observação caso esta tenha sido adicionada com sucesso	
installThingGateway	name description content	Possibilita ao gestor do sistema adicionar um novo gateway para comunicar com redes de sensores ou dispositivos maneira a comunicar com diferentes tecnologias e permitir a expansão do sistema e adaptabilidade a novas tecnologias. Este serviço encontra-se implementado no middleware, contudo a aplicação desenvolvida não tem suporte para envio dos gateways.
	identificador do gateway caso este tenha sido instalado corretamente	
instantiateService	instanceId serviceMetadata	Instancia um serviço instalado anteriormente. No contexto da aplicação desenvolvida é possível selecionar qual o serviço que se pretende inicializar a partir do conjunto de todos os módulos instalados, bem como selecionar a partir de uma lista todos aqueles que fazem parte das dependências do módulo que se está a iniciar.
	identificador da instância do serviço, caso esta tenha sido instanciada com sucesso	

deployThingsCode	thingGatewayID content	Embora esteja implementado no middleware, o sistema final não faz uso deste serviço. Caso os dispositivos da rede de sensores suportem ser reprogramados remotamente, e o respetivo gateway permita esteja preparado para tal, através deste serviço é possível reprogramar dispositivos da rede de sensores.
	"true" caso o código tenha sido enviado com sucesso, "false" caso contrário	
readObservation	getObservationXML	Serviço essencial para qual middleware de suporte a redes de sensores. Permite requisitar observações ao sistema a partir de um determinado conjunto de restrições. É fortemente utilizada na aplicação desenvolvida.
	observação, ou conjunto de observações em XML	
subscribeService	instanceId serviceId callbackRef	Subscreve um serviço de modo a poder receber dados ou notificações do mesmo. Embora implementada no middleware esta funcionalidade não é utilizado na aplicação.
	"true" caso o serviço tenha sido subscrito, "false" caso contrário	
unsubscribeService	instanceId serviceId callbackRef	De modo a reverter as alterações efectuadas pelo serviço anterior é possível cancelar aos subscrições a serviços.
	"true" caso a subscrição tenha sido cancelada, "false" caso contrário	
alterDateTime	strDateTime	Serviço que interage com o sistema operativo atualizando o mesmo para a data e hora recebidas. Este serviço é apresentado na aplicação através da opção de "Sincronizar".
	"true" caso a data do sistema tenha sido alterada, "false" caso contrário	

Figura E.1: Serviços *web* disponibilizados pelo *middleware*

Apêndice F

Tabelas com valores recolhidos nos testes

De maneira a complementar as informações apresentadas no capítulo reservado aos resultados, são aqui apresentadas as tabelas com os os valores recolhidos nos testes.

F.1 Teste de comunicação sob influência dos materiais isolantes

Distâncias	**Mensagens recebidas por minuto**	
	Dispositivo isolado com caixa de plástico	**Dispositivo isolado com caixa de plástico + fire shelter (alumínio + fibra de vidro)**
20	37	31
40	37	42
60	26	40
80	41	42
100	33	31
120	42	35
140	35	37

Tabela F.1: Numero de mensagens recebidas por minuto sob influência dos isolantes

F.2 Teste de temperatura ao material isolante (*fire shelter*)

	Dispositivo com sensor isolado	**Dispositivo com sensor não isolado**
Quantidade de amostras	154	71
Máximo (ºC)	70.3	147.53
Mínimo (ºC)	22.9	23.4
Média (ºC)	37.77	56.9
Mediana (ºC)	28.8	36.6
Moda (ºC)	22.9	29.3

Tabela F.2: Resumo de dados de temperaturas dentro e fora do *fire shelter*

F.3 Teste ao *middleware* com número fixo de observações

Pedido	Número de observações	Tempo (milisegundos)
1	20	76128
2	20	16496
3	20	14686
4	20	12087
5	20	11740
6	20	11289
7	20	11171
8	20	10994
9	20	11225
10	20	11181
11	20	11276
12	20	11937
13	20	10216
14	20	10567
15	20	10229
16	20	10538
17	20	10161
18	20	10393
19	20	10355
20	20	10296
21	20	10358
22	20	10346
23	20	10306
24	20	10406
25	20	10355

Tabela F.3: Resultados do teste ao *middleware* com número fixo de observações

F.4 Teste ao *middleware* com variação do número de observações

Pedido	Número de observações	Tempo (milisegundos)
1	2	3465
2	4	5984
3	6	7626
4	8	9681
5	10	13796
6	12	13204
7	14	17979
8	16	20166
9	18	18462
10	20	21057
11	22	21918
12	24	24361
13	26	26262
14	28	28115
15	30	30952
16	32	35946
17	34	34835
18	36	37243
19	38	39550
20	40	41365
21	42	44404
22	44	46188
23	46	49177
24	48	52279
25	50	54167

Tabela F.4: Resultados do teste ao *middleware* com variação do número de observações

F.5 Teste ao *middleware* com variação do número de utilizadores

Pedido/Nº utilizadores	**Tempo mínimo**	**Tempo máximo**	**Tempo médio**	**Tempo último pedido**	**Memória**
1	11609	11609	11609,000	11609	52,10%
2	21479	21689	21584,000	21689	54,40%
3	31094	31838	31470,330	31838	55,20%
4	39302	40783	40101,000	40783	56,60%
5	49148	53688	53073,199	53688	59,20%
6	55678	60859	58663,328	60859	63,10%
7	60118	70280	67212,852	69983	66,80%
8	77465	91084	83209,000	91084	67,30%
9	115694	137248	128587,453	137248	67,30%
10	151106	179050	167147,203	179050	67,30%
11	ERRO: ”OutOfMemory”				

Tabela F.5: Resultados do teste ao *middleware* com variação do número de utilizadores

Os valores da coluna “Memória” apresentam a percentagem de memória do dispositivo ocupada pelo processo correspondente ao *middleware*, e na coluna “Tempo médio” é feita a média dos tempos obtidos em cada *thread*.

Bibliografia

[1] Arduino UNO. http://arduino.cc/en/Main/arduinoBoardUno.

[2] Debian Wheezy + JavaSE. https://wiki.debian.org/Java/Sun.

[3] From Arduino to a Microcontroller on a Breadboard. http://arduino.cc/en/Tutorial/ArduinoToBreadboard.

[4] Introduction to SensorML. http://www.ogcnetwork.net/SensorML_Intro.

[5] JavaME. http://www.oracle.com/technetwork/java/embedded/downloads/javame/index.html.

[6] JavaSE. https://www.java.com/en/download/help/sysreq.xml.

[7] Python. http://www.python.org/about/.

[8] Raspberry PI Model B. http://raspberrypi.org.

[9] Raspberry PI Software. http://www.raspberrypi.org/faqs#softwareOS.

[10] Raspbian. http://www.raspbian.org/FrontPage.

[11] Sensor Planning Service. http://www.opengeospatial.org/standards/sps.

[12] Transducer Markup Language. http://www.opengeospatial.org/standards/tml.

[13] Intoxicación por monóxido de carbono, 2000. http://www.saludalia.com/urgencias/intoxicacion-por-monoxido-de-carbono-co.

[14] Karl Aberer, Manfred Hauswirth, and Ali Salehi. The global sensor networks middleware for eficient and fexible deployment and interconnection of sensor networks. Technical Report LSIR-REPORT-2006-006, Ecole Polytechnique Fédérale de Lausanne, 2006.

[15] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Middleware support for the "Internet of Things", 2006.

[16] Karl Aberer and Ali Salehi. Global Sensor Network, 2004. `http://sourceforge.net/apps/trac/gsn`.

[17] Mike Botts, George Percivall, Carl Reed, and John Davidson. OGC Sensor Web Enablement: Overview and high level architecture, 2007.

[18] Arne Broering, Theodor Forester, Carsten Priess, and Simon Jirka. Sensor bus: An intermediary layer for linking geosensors and the sensor web. In *1st International Conference on Computing for Geospatial Research and Applications (COM. Geo)*, pages 21–30. ACM New York, 2010.

[19] Christopher Date. *SQL & Relational Theory*. O'Reilly Media, 2009.

[20] Eusébio Z. E. Conceição, Maria Manuela J. R. Lúcio, and Domingos X. Viegas. Numerical simulation of the thermal sensation of a fireman equipped with special clothing. In *Forest Ecology and Management*, volume 234. ELSEVIER, 2006.

[21] Simon Cox. Observations and Measurements - XML Implementation Standard, 2011.

[22] Simon Cox. OGC Abstract Specification: Geographic information — Observations and measurements, 2011.

[23] Diogo José Fernandes da Cunha. Sistem de detecção e envio de alarmes. Master's thesis, Universidade de Aveiro, 2010.

[24] Chinthaka Dissanayake, Malka Halgamuge, Kotagiri Ramomohanarao, Bill Moran, and Peter Farrell. The Signal Propagation Effects in IEEE 802.15.4 Radio Link in Fire Environment. In *Information and Automation for Sustainability (ICIAFs)*, pages 411–414. IEEE Computer Society, 2010.

[25] Robert Faludi. *Building Wireless Sensor Networks*. O'Reilly Media, 2010.

[26] Apache Software Foundation. Apache Cordova, 2014. `http://cordova.apache.org/`.

[27] Martin Fowler. *Patterns of Enterprise*.

[28] Martin Fowler. *Analysis patterns: Reusable object models*. Addison-Wesley, 1997.

[29] Scott Health and Safety Ltd. Input control management system for displaying a status of firefighters on an electronic control panel and a digital pressure gauge, 2013.

[30] IEEE. IEEE 1451 family - Standard for a Smart Transducer Interface for Sensors and Actuators, 2000.

[31] jQuery Foundation. JQuery Mobile, 2014. http://jquerymobile.com/.

[32] Libelium. Documentation: GPS Module for Raspberry Pi Tutorial, 2013. http://www.cooking-hacks.com/documentation/tutorials/raspberry-pi-gps.

[33] Ana Isabel Miranda, Vera Martins, Pedro Cascão, Jorge Humberto Amorim, Joana Valente, Richard Tavares, Carlos Borrego, Oxana Tchepel, António Jorge Ferreira, Carlos Robalo Cordeiro, Domingos Xavier Viegas, Luís Mário Ribeiro, and Luís Paulo Pita. Monitoring Firefighters Exposure to Smoke During Fire Experiments and Wildfires. *Environment International*, 36(7):736–745, 2010.

[34] Nina Subbotina. Secuelas neurológicas post intoxicación por Monóxido de Carbono. http://www.hipercamaras.com.ar/intoxicacion-por-monoxido-de-cabono.htm.

[35] Cuno Pfister. *Getting Started with The Internet of Things*. O'Reilly Media, 2001.

[36] Rui José Laranjeira Pires. Programação homogénea de redes de sensores usando o middlware MuFFIN. Master's thesis, Faculdade de Ciências da Universidade de Lisboa, 2013.

[37] A. M. Raimundo and A. R. Figueiredo. Human thermophisiological response to high intensity radiation fluxes near a forest fire line. In *Forest Ecology and Management*, volume 234, 2006.

[38] Hélder Silva. *Ambiente Térmico e Ventilação*. Edições Silabo, 2013.

[39] Lübeck Stefan Barten, Stockelsdorf Volker Kuhn, and Krummesse Andreas Suerig. Rechargeable gas-measuring system, 2013.

[40] Bruno Alexandre Loureiro Valente. Um middlware para A Internet das Coisas. Master's thesis, Faculdade de Ciências da Universidade de Lisboa, 2011.

[41] Stefan von Gagern. Native vs. web app vs. hybrid – what is the optimal developer strategy?, 2013. http://www.developergarden.com/en/blog/articles/article/native-vs-web-app-vs-hybrid-what-is-the-optimal-developer-strategy/